
Administering Platform™ LSF™

Version 7 Update 2
Release date: November 2007
Last modified: January 31, 2008
Comments to: doc@platform.com
Support: support@platform.com



Copyright

© 1994-2008, Platform Computing Inc.

Although the information in this document has been carefully reviewed, Platform Computing Inc. (“Platform”) does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED “AS IS” AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

We’d like to hear from you

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Document redistribution and translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole.

Internal redistribution

You may only redistribute this document internally within your organization (for example, on an intranet) provided that you continue to check the Platform Web site for updates and update your version of the documentation. You may not make it available to your organization over the Internet.

Trademarks

LSF is a registered trademark of Platform Computing Inc. in the United States and in other jurisdictions.

POWERING HIGH PERFORMANCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOBSCHEDULER, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Inc. in the United States and in other jurisdictions.

UNIX is a registered trademark of The Open Group in the United States and in other jurisdictions.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective owners.

Third-party license agreements

<http://www.platform.com/Company/third.part.license.htm>

Third-party copyright notices

<http://www.platform.com/Company/Third.Party.Copyright.htm>

Contents

1	About Platform LSF	13
	Contents	13
	Learn about Platform LSF	14
	Cluster Concepts	14
	Job Life Cycle	25
2	How the System Works	27
	Contents	27
	Job Submission	27
	Job Scheduling and Dispatch	29
	Host Selection	31
	Job Execution Environment	32
	Fault Tolerance	33

Part I: Managing Your Cluster

3	Working with Your Cluster	37
	Contents	37
	Viewing cluster information	38
	Example directory structures	43
	Cluster administrators	45
	Controlling daemons	46
	Controlling mbatchd	48
	Customize batch command messages	50
	Reconfiguring your cluster	51
4	Working with Hosts	53
	Contents	53
	Host status	53
	How LIM Determines Host Models and Types	55
	Viewing Host Information	56
	Controlling Hosts	61
	Adding a Host	63
	Remove a Host	66
	Adding Hosts Dynamically	66
	Add Host Types and Host Models to Isf.shared	74
	Registering Service Ports	75
	Host Naming	77

Contents

	Hosts with Multiple Addresses	79
	Using IPv6 Addresses	82
	Host Groups	83
	Tuning CPU Factors	86
	Handling Host-level Job Exceptions	88
5	Working with Queues	91
	Contents	91
	Queue States	91
	Viewing Queue Information	92
	Control Queues	94
	Add and Remove Queues	97
	Manage Queues	99
	Handling Job Exceptions in Queues	100
6	Managing Jobs	103
	Contents	103
	Understanding Job States	103
	View Job Information	106
	Changing Job Order Within Queues	109
	Switch Jobs from One Queue to Another	110
	Forcing Job Execution	111
	Suspending and Resuming Jobs	111
	Killing Jobs	112
	Sending a Signal to a Job	113
	Using Job Groups	114
	Handling Job Exceptions	123
7	Managing Users and User Groups	129
	Contents	129
	Viewing User and User Group Information	129
	About User Groups	131
	Existing User Groups as LSF User Groups	131
	LSF User Groups	132
8	Platform LSF Licensing	135
	Contents	135
	The LSF License File	136
	How LSF Permanent Licensing Works	140
	Installing a Demo License	142
	Installing a Permanent License	144
	Updating a License	149
	FLEXlm Basics	151
	Multiple FLEXlm License Server Hosts	154
	Partial Licensing	156
	Floating Client Licenses	160
	Troubleshooting License Issues	166

9	Managing LSF on Platform EGO	169
	Contents	169
	About LSF on Platform EGO	170
	LSF and EGO directory structure	173
	Configuring LSF and EGO	177
	Managing LSF daemons through EGO	181
	EGO Administrative Basics	184
	Logging and troubleshooting	201
	Frequently asked questions	209
10	Cluster Version Management and Patching on UNIX and Linux	211
	Contents	211
	Scope	212
	Patch installation interaction diagram	213
	Patch rollback interaction diagram	214
	Version management components	215
	Version management concepts	217
	Cluster patching behavior table	219
	Cluster rollback behavior table	220
	Version management files	220
	Version management commands	221
	Installing update releases on UNIX and Linux	222
	Installing fixes on UNIX and Linux	223
	Rolling back patches on UNIX and Linux	223
	Patching the Oracle database	224
	Patching the Derby database	225

Part II: Working with Resources

11	Understanding Resources	229
	Contents	229
	About LSF Resources	230
	How Resources are Classified	232
	How LSF Uses Resources	235
	Load Indices	237
	Static Resources	240
	Automatic Detection of Hardware Reconfiguration	246
12	Adding Resources	249
	Contents	249
	About Configured Resources	249
	Add New Resources to Your Cluster	249
	Configuring Isf.shared Resource Section	250
	Configuring Isf.cluster.cluster_name ResourceMap Section	251
	Static Shared Resource Reservation	253
	External Load Indices and ELIM	253
	Modifying a Built-In Load Index	258

Contents

13	Managing Software Licenses with LSF	261
	Contents	261
	Using Licensed Software with LSF	261
	Host-locked Licenses	261
	Counted Host-Locked Licenses	261
	Network Floating Licenses	262

Part III: Job Scheduling Policies

14	Time Syntax and Configuration	269
	Contents	269
	Specifying Time Values	269
	Specifying Time Windows	269
	Specifying Time Expressions	270
	Using Automatic Time-based Configuration	271
15	Deadline Constraint and Exclusive Scheduling	275
	Contents	275
	Using Deadline Constraint Scheduling	275
	Using Exclusive Scheduling	276
16	Preemptive Scheduling	277
	Contents	277
	About Preemptive Scheduling	277
17	Specifying Resource Requirements	279
	Contents	279
	About Resource Requirements	279
	Queue-level Resource Requirements	280
	Job-level Resource Requirements	281
	About Resource Requirement Strings	282
	Selection String	284
	Order String	287
	Usage String	287
	Span String	291
	Same String	292
18	Fairshare Scheduling	295
	Contents	295
	Understanding Fairshare Scheduling	296
	User Share Assignments	297
	Dynamic User Priority	298
	How Fairshare Affects Job Dispatch Order	300
	Host Partition User-based Fairshare	301
	Queue-level User-based Fairshare	302
	Cross-queue User-based Fairshare	302
	Hierarchical User-based Fairshare	306
	Queue-based Fairshare	309
	Configuring Slot Allocation per Queue	311

	View Queue-based Fairshare Allocations	313
	Typical Slot Allocation Scenarios	314
	Using Historical and Committed Run Time	319
	Users Affected by Multiple Fairshare Policies	323
	Ways to Configure Fairshare	324
19	Goal-Oriented SLA-Driven Scheduling	327
	Contents	327
	Using Goal-Oriented SLA Scheduling	327
	Configuring Service Classes for SLA Scheduling	330
	View Information about SLAs and Service Classes	332
	Understanding Service Class Behavior	336
	EGO-enabled SLA scheduling	341
<hr/>		
Part IV: Job Scheduling and Dispatch		
20	Working with Application Profiles	357
	Contents	357
	Manage application profiles	357
	Use application profiles	359
	View application profile information	360
	How application profiles interact with queue and job parameters	362
21	Resource Allocation Limits	369
	Contents	369
	About Resource Allocation Limits	370
	Configuring Resource Allocation Limits	375
	Viewing Information about Resource Allocation Limits	384
22	Reserving Resources	387
	Contents	387
	About Resource Reservation	387
	Using Resource Reservation	388
	Memory Reservation for Pending Jobs	390
	Time-based Slot Reservation	392
	Viewing Resource Reservation Information	399
23	Advance Reservation	403
	Contents	403
	Understanding Advance Reservations	404
	Configure Advance Reservation	405
	Using Advance Reservation	407
24	Dispatch and Run Windows	419
	Contents	419
	Dispatch and Run Windows	419
	Run Windows	419
	Dispatch Windows	420

Contents

25	Job Dependencies	423
	Contents	423
	Job Dependency Scheduling	423
	Dependency Conditions	424
26	Job Priorities	429
	Contents	429
	User-Assigned Job Priority	429
	Automatic Job Priority Escalation	431
	Absolute Job Priority Scheduling	431
27	Job Requeue and Job Rerun	441
	Contents	441
	About Job Requeue	441
	Automatic Job Requeue	441
	Reverse Requeue	442
	Exclusive Job Requeue	442
	User-Specified Job Requeue	443
	Automatic Job Rerun	444
28	Job Checkpoint, Restart, and Migration	447
	Contents	447
	Checkpoint and restart options	447
	Checkpoint directory and files	448
	Checkpoint and restart executables	449
	Job restart	449
	Job migration	450
29	Chunk Job Dispatch	453
	Contents	453
	About Job Chunking	453
	Configure Chunk Job Dispatch	454
	Submitting and Controlling Chunk Jobs	456
30	Job Arrays	459
	Contents	459
	Create a Job Array	459
	Handling Input and Output Files	461
	Redirecting Standard Input and Output	461
	Passing Arguments on the Command Line	462
	Job Array Dependencies	463
	Monitoring Job Arrays	463
	Individual job status	464
	Specific job status	465
	Controlling Job Arrays	465
	Requeuing a Job Array	466
	Job Array Job Slot Limit	467

31	Running Parallel Jobs	469
	Contents	469
	How LSF Runs Parallel Jobs	469
	Preparing Your Environment to Submit Parallel Jobs to LSF	470
	Submitting Parallel Jobs	470
	Starting Parallel Tasks with LSF Utilities	471
	Job Slot Limits For Parallel Jobs	472
	Specifying a Minimum and Maximum Number of Processors	472
	Specifying a First Execution Host	473
	Controlling Processor Allocation Across Hosts	474
	Running Parallel Processes on Homogeneous Hosts	477
	Limiting the Number of Processors Allocated	478
	Reserving Processors	481
	Reserving Memory for Pending Parallel Jobs	482
	Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots	483
	Parallel Fairshare	492
	How Deadline Constraint Scheduling Works For Parallel Jobs	493
	Optimized Preemption of Parallel Jobs	493
32	Submitting Jobs Using JSDL	495
	Contents	495
	Why Use JSDL?	495
	Using JSDL Files with LSF	495
	Collecting resource values using elim.jsdl	504
<hr/>		
Part V: Controlling Job Execution		
33	Runtime Resource Usage Limits	509
	Contents	509
	About Resource Usage Limits	509
	Specifying Resource Usage Limits	513
	Supported Resource Usage Limits and Syntax	515
	Examples	520
	CPU Time and Run Time Normalization	521
34	Load Thresholds	523
	Contents	523
	Automatic Job Suspension	523
	Suspending Conditions	524
35	Pre-Execution and Post-Execution Commands	529
	Contents	529
	About Pre-Execution and Post-Execution Commands	529
	Configuring Pre- and Post-Execution Commands	531
36	Job Starters	537
	Contents	537
	About Job Starters	537
	Command-Level Job Starters	538

Contents

- Queue-Level Job Starters 540
- Controlling Execution Environment Using Job Starters 541
- 37 External Job Submission and Execution Controls 543
 - Contents 543
 - Understanding External Executables 543
 - Using esub 544
 - Existing esub 552
 - Working with eexec 552
- 38 Configuring Job Controls 553
 - Contents 553
 - Default Job Control Actions 553
 - Configuring Job Control Actions 555
 - Customizing Cross-Platform Signal Conversion 558

Part VI: Interactive Jobs

- 39 Interactive Jobs with bsub 561
 - Contents 561
 - About Interactive Jobs 561
 - Submitting Interactive Jobs 562
 - Performance Tuning for Interactive Batch Jobs 564
 - Interactive Batch Job Messaging 567
 - Running X Applications with bsub 568
 - Writing Job Scripts 568
 - Registering utmp File Entries for Interactive Batch Jobs 571
- 40 Running Interactive and Remote Tasks 573
 - Contents 573
 - Running Remote Tasks 573
 - Interactive Tasks 576
 - Load Sharing Interactive Sessions 578
 - Load Sharing X Applications 578

Part VII: Monitoring Your Cluster

- 41 Achieving Performance and Scalability 583
 - Contents 583
 - Optimizing Performance in Large Sites 583
 - Tuning UNIX for Large Clusters 584
 - Tuning LSF for Large Clusters 585
 - Monitoring Performance Metrics in Real Time 593
- 42 Reporting 597
 - Contents 597
 - Introduction to Reporting 597
 - Getting Started with Standard Reports 598

	Custom Reports	600
	System Description	604
	Reports Administration	606
	Test the Reporting Feature	617
	Disable the Reporting Feature	618
	Move to a Production Database	619
43	Event Generation	623
	Contents	623
	Event Generation	623
	Enabling event generation	623
	Events list	624
	Arguments passed to the LSF event program	624
44	Tuning the Cluster	627
	Contents	627
	Tuning LIM	628
	Improving performance of mbatchd query requests on UNIX	635
45	Authentication and Authorization	641
	Contents	641
	Authentication options	641
	Authorization options	643
46	Job Email and Job File Spooling	649
	Contents	649
	Mail Notification When a Job Starts	649
	File Spooling for Job Input, Output, and Command Files	652
	Specifying a job command file (bsub -Zs)	653
	About the job spooling directory (JOB_SPOOL_DIR)	653
	Modifying the job input file	654
	Modifying the job command file	654
	For more information	654
47	Non-Shared File Systems	655
	Contents	655
	About Directories and Files	655
	Using LSF with Non-Shared File Systems	656
	Remote File Access	656
	File Transfer Mechanism (lsrctp)	658
48	Error and Event Logging	661
	Contents	661
	System Directories and Log Files	661
	Managing Error Logs	662
	System Event Log	663
	Duplicate Logging of Event Logs	664
	LSF Job Termination Reason Logging	665

Contents

49	Troubleshooting and Error Messages	671
	Contents	671
	Shared File Access	672
	Common LSF Problems	673
	Error Messages	680
	Setting Daemon Message Log to Debug Level	687
	Setting Daemon Timing Levels	690
50	Understanding Platform LSF Job Exit Information	693
	Contents	693
	Why did my job exit?	694
	How LSF translates events into exit codes	694
	Application and system exit values	695
	LSF job termination reason logging	697
	Job termination by LSF exit information	701
	LSF RMS integration exit values	703

Part VIII: LSF Utilities

51	Using Istcsh	707
	Contents	707
	About Istcsh	707
	Differences from Other Shells	709
	Limitations	709
	Starting Istcsh	710
	Using Istcsh as Your Login Shell	710
	Host Redirection	711
	Task Control	712
	Built-in Commands	712
	Writing Shell Scripts in Istcsh	714
	Index	717

About Platform LSF

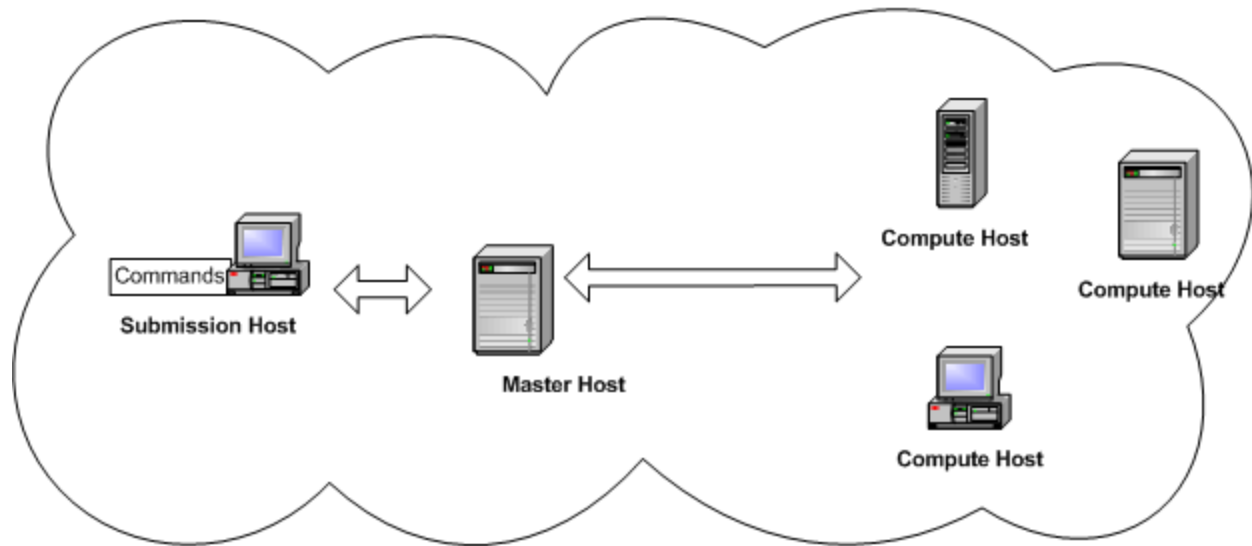
Contents

- ◆ [Learn about Platform LSF](#) on page 14
- ◆ [Cluster Concepts](#) on page 14
- ◆ [Job Life Cycle](#) on page 25

Learn about Platform LSF

Before using Platform LSF for the first time, you should download and read LSF Version 7 Release Notes for the latest information about what's new in the current release and other important information.

Cluster Concepts



Clusters, jobs, and queues

Cluster

A group of computers (hosts) running LSF that work together as a single unit, combining computing power and sharing workload and resources. A cluster provides a single-system image for disparate computing resources.

Hosts can be grouped into clusters in a number of ways. A cluster could contain:

- ◆ All the hosts in a single administrative group
- ◆ All the hosts on one file server or sub-network
- ◆ Hosts that perform similar functions

Commands:

- ◆ `lshosts`—View static resource information about hosts in the cluster
- ◆ `bhosts`—View resource and job information about server hosts in the cluster
- ◆ `lsid`—View the cluster name
- ◆ `lsclusters`—View cluster status and size

Configuration:

- ◆ Define hosts in your cluster in `lsf.cluster.cluster_name`

TIP: The name of your cluster should be unique. It should not be the same as any host or queue.

Job

A unit of work run in the LSF system. A job is a command submitted to LSF for execution. LSF schedules, controls, and tracks the job according to configured policies.

Jobs can be complex problems, simulation scenarios, extensive calculations, anything that needs compute power.

Commands:

- ◆ `bjobs` — View jobs in the system
- ◆ `bsub` — Submit jobs

Job slot

A job slot is a bucket into which a single unit of work is assigned in the LSF system. Hosts are configured to have a number of job slots available and queues dispatch jobs to fill job slots.

Commands:

- ◆ `bhosts` — View job slot limits for hosts and host groups
- ◆ `bqueues` — View job slot limits for queues
- ◆ `busers` — View job slot limits for users and user groups

Configuration:

- ◆ Define job slot limits in `lsb.resources`.

Job states

LSF jobs have the following states:

- ◆ `PEND` — Waiting in a queue for scheduling and dispatch
- ◆ `RUN` — Dispatched to a host and running
- ◆ `DONE` — Finished normally with zero exit value
- ◆ `EXITED` — Finished with non-zero exit value
- ◆ `PSUSP` — Suspended while pending
- ◆ `USUSP` — Suspended by user
- ◆ `SSUSP` — Suspended by the LSF system
- ◆ `POST_DONE` — Post-processing completed without errors
- ◆ `POST_ERR` — Post-processing completed with errors
- ◆ `WAIT` — Members of a chunk job that are waiting to run

Queue

A clusterwide container for jobs. All jobs wait in queues until they are scheduled and dispatched to hosts.

Queues do not correspond to individual hosts; each queue can use all server hosts in the cluster, or a configured subset of the server hosts.

When you submit a job to a queue, you do not need to specify an execution host. LSF dispatches the job to the best available execution host in the cluster to run that job.

Queues implement different job scheduling and control policies.

Commands:

- ◆ `bqueues` — View available queues
- ◆ `bsub -q` — Submit a job to a specific queue

Cluster Concepts

- ◆ `bparams` — View default queues

Configuration:

- ◆ Define queues in `lsb.queues`

TIP: The names of your queues should be unique. They should not be the same as the cluster name or any host in the cluster.

First-come, first-served (FCFS) scheduling

The default type of scheduling in LSF. Jobs are considered for dispatch based on their order in the queue.

Hosts

Host

An individual computer in the cluster.

Each host may have more than 1 processor. Multiprocessor hosts are used to run parallel jobs. A multiprocessor host with a single process queue is considered a single machine, while a box full of processors that each have their own process queue is treated as a group of separate machines.

Commands:

- ◆ `lsload` — View load on hosts
- ◆ `lshosts` — View configuration information about hosts in the cluster including number of CPUs, model, type, and whether the host is a client or server
- ◆ `bhosts` — View batch server hosts in the cluster

TIP: The names of your hosts should be unique. They should not be the same as the cluster name or any queue defined for the cluster.

Submission host

The host where jobs are submitted to the cluster.

Jobs are submitted using the `bsub` command or from an application that uses the LSF API.

Client hosts and server hosts can act as submission hosts.

Commands:

- ◆ `bsub` — Submit a job
- ◆ `bjobs` — View jobs that are submitted

Execution host

The host where a job runs. Can be the same as the submission host. All execution hosts are server hosts.

Commands:

- ◆ `bjobs` — View where a job runs

Server host

Hosts that are capable of submitting and executing jobs. A server host runs `sbatchd` to execute server requests and apply local policies.

Commands:

- ◆ `lshosts` — View hosts that are servers (`server=Yes`)

Configuration:

- ◆ Server hosts are defined in the `lsf.cluster.cluster_name` file by setting the value of `server` to 1

Client host

Hosts that are only capable of submitting jobs to the cluster. Client hosts run LSF commands and act only as submission hosts. Client hosts do not execute jobs or run LSF daemons.

Commands:

- ◆ `lshosts` — View hosts that are clients (`server=No`)

Configuration:

- ◆ Client hosts are defined in the `lsf.cluster.cluster_name` file by setting the value of `server` to 0

Master host

Where the master LIM and `mbatchd` run. An LSF server host that acts as the overall coordinator for that cluster. Each cluster has one master host to do all job scheduling and dispatch. If the master host goes down, another LSF server in the cluster becomes the master host.

All LSF daemons run on the master host. The LIM on the master host is the master LIM.

Commands:

- ◆ `lsid` — View the master host name

Configuration:

- ◆ The master host is defined along with other candidate master hosts by `LSF_MASTER_LIST` in `lsf.conf`.

LSF daemons

LSF daemon	Role
<code>mbatchd</code>	Job requests and dispatch
<code>mbschd</code>	Job scheduling
<code>sbatchd</code>	Job execution
<code>res</code>	
<code>lim</code>	Host information
<code>pim</code>	Job process information
<code>elim</code>	Collect and track custom dynamic load indices

`mbatchd`

Master Batch Daemon running on the master host. Started by `sbatchd`. Responsible for the overall state of jobs in the system.

Receives job submission, and information query requests. Manages jobs held in queues. Dispatches jobs to hosts as determined by `mbschd`.

Configuration:

- ◆ Port number defined in `lsf.conf`.

`mbschd`

Master Batch Scheduler Daemon running on the master host. Works with `mbatchd`. Started by `mbatchd`.

Makes scheduling decisions based on job requirements and policies.

sbatchd

Slave Batch Daemon running on each server host. Receives the request to run the job from `mbatchd` and manages local execution of the job. Responsible for enforcing local policies and maintaining the state of jobs on the host.

`sbatchd` forks a child `sbatchd` for every job. The child `sbatchd` runs an instance of `res` to create the execution environment in which the job runs. The child `sbatchd` exits when the job is complete.

Commands:

- ◆ `badmin hstartup`— Starts `sbatchd`
- ◆ `badmin hshutdown`— Shuts down `sbatchd`
- ◆ `badmin hrestart`— Restarts `sbatchd`

Configuration:

- ◆ Port number defined in `lsf.conf`

res

Remote Execution Server (RES) running on each server host. Accepts remote execution requests to provide transparent and secure remote execution of jobs and tasks.

Commands:

- ◆ `lsadmin resstartup`— Starts `res`
- ◆ `lsadmin resshutdown`— Shuts down `res`
- ◆ `lsadmin resrestart`— Restarts `res`

Configuration:

- ◆ Port number defined in `lsf.conf`

lim

Load Information Manager (LIM) running on each server host. Collects host load and configuration information and forwards it to the master LIM running on the master host. Reports the information displayed by `lsload` and `lshosts`.

Static indices are reported when the LIM starts up or when the number of CPUs (`ncpus`) change. Static indices are:

- ◆ Number of CPUs (`ncpus`)
- ◆ Number of disks (`ndisks`)
- ◆ Total available memory (`maxmem`)
- ◆ Total available swap (`maxswp`)
- ◆ Total available temp (`maxtmp`)
- ◆ The number of physical processors per CPU configured on a host (`nprocs`).
- ◆ The number of cores per processor configured on a host (`ncores`).
- ◆ The number of threads per core configured on a host (`nthreads`).

Dynamic indices for host load collected at regular intervals are:

- ◆ Hosts status (`status`)
- ◆ 15 second, 1 minute, and 15 minute run queue lengths (`r15s`, `r1m`, and `r15m`)
- ◆ CPU utilization (`ut`)

- ◆ Paging rate (`pg`)
- ◆ Number of login sessions (`ls`)
- ◆ Interactive idle time (`it`)
- ◆ Available swap space (`swp`)
- ◆ Available memory (`mem`)
- ◆ Available temp space (`tmp`)
- ◆ Disk IO rate (`io`)

Commands:

- ◆ `lsadmin limstartup`— Starts LIM
- ◆ `lsadmin limshutdown`— Shuts down LIM
- ◆ `lsadmin limrestart`— Restarts LIM
- ◆ `lsload`— View dynamic load values
- ◆ `lshosts`— View static host load values

Configuration:

- ◆ Port number defined in `lsf.conf`.

Master LIM

The LIM running on the master host. Receives load information from the LIMs running on hosts in the cluster.

Forwards load information to `mbatchd`, which forwards this information to `mbschd` to support scheduling decisions. If the master LIM becomes unavailable, a LIM on another host automatically takes over.

Commands:

- ◆ `lsadmin limstartup`— Starts LIM
- ◆ `lsadmin limshutdown`— Shuts down LIM
- ◆ `lsadmin limrestart`— Restarts LIM
- ◆ `lsload`— View dynamic load values
- ◆ `lshosts`— View static host load values

Configuration:

- ◆ Port number defined in `lsf.conf`.

ELIM

External LIM (ELIM) is a site-definable executable that collects and tracks custom dynamic load indices. An ELIM can be a shell script or a compiled binary program, which returns the values of the dynamic resources you define. The ELIM executable must be named `elim` and located in `LSF_SERVERDIR`.

pim

Process Information Manager (PIM) running on each server host. Started by LIM, which periodically checks on PIM and restarts it if it dies.

Collects information about job processes running on the host such as CPU and memory used by the job, and reports the information to `sbatchd`.

Commands:

- ◆ `bjobs`— View job information

Batch jobs and tasks

You can either run jobs through the batch system where jobs are held in queues, or you can interactively run tasks without going through the batch system, such as tests for example.

Job

A unit of work run in the LSF system. A job is a command submitted to LSF for execution, using the `bsub` command. LSF schedules, controls, and tracks the job according to configured policies.

Jobs can be complex problems, simulation scenarios, extensive calculations, anything that needs compute power.

Commands:

- ◆ `bjobs`—View jobs in the system
- ◆ `bsub`—Submit jobs

Interactive batch job

A batch job that allows you to interact with the application and still take advantage of LSF scheduling policies and fault tolerance. All input and output are through the terminal that you used to type the job submission command.

When you submit an interactive job, a message is displayed while the job is awaiting scheduling. A new job cannot be submitted until the interactive job is completed or terminated.

The `bsub` command stops display of output from the shell until the job completes, and no mail is sent to you by default. Use `Ctrl-C` at any time to terminate the job.

Commands:

- ◆ `bsub -I`—Submit an interactive job

Interactive task

A command that is not submitted to a batch queue and scheduled by LSF, but is dispatched immediately. LSF locates the resources needed by the task and chooses the best host among the candidate hosts that has the required resources and is lightly loaded. Each command can be a single process, or it can be a group of cooperating processes.

Tasks are run without using the batch processing features of LSF but still with the advantage of resource requirements and selection of the best host to run the task based on load.

Commands:

- ◆ `lrun`—Submit an interactive task
- ◆ `lsgun`—Submit an interactive task to a group of hosts
- ◆ See also LSF utilities such as `ch`, `lsacct`, `lsacctmrg`, `lslogin`, `lsplace`, `lsload`, `lsloadadj`, `lselectible`, `lsmn`, `lstcsh`

Local task

An application or command that does not make sense to run remotely. For example, the `ls` command on UNIX.

Commands:

- ◆ `lsltasks`—View and add tasks

Configuration:

- ◆ `lsf.task`—Configure systemwide resource requirements for tasks

- ◆ `lsf.task.cluster`—Configure clusterwide resource requirements for tasks
- ◆ `.lsftask`—Configure user-specific tasks

Remote task

An application or command that can be run on another machine in the cluster.

Commands:

- ◆ `lsrtasks`—View and add tasks

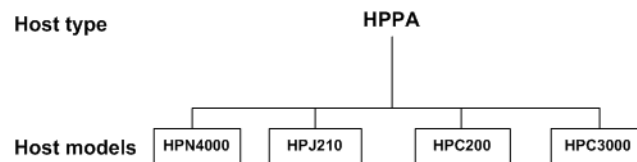
Configuration:

- ◆ `lsf.task`—Configure systemwide resource requirements for tasks
- ◆ `lsf.task.cluster`—Configure clusterwide resource requirements for tasks
- ◆ `.lsftask`—Configure user-specific tasks

Host types and host models

Hosts in LSF are characterized by host type and host model.

The following example has HP hosts. The host type is HPPA. Host models can be HPN4000, HPJ210, etc.



Host type

The combination of operating system version and host CPU architecture.

All computers that run the same operating system on the same computer architecture are of the same type—in other words, binary-compatible with each other.

Each host type usually requires a different set of LSF binary files.

Commands:

- ◆ `lsinfo -t`—View all host types defined in `lsf.shared`

Configuration:

- ◆ Defined in `lsf.shared`
- ◆ Mapped to hosts in `lsf.cluster.cluster_name`

Host model

The combination of host type and CPU speed (CPU factor) of the computer.

All hosts of the same relative speed are assigned the same host model.

The CPU factor is taken into consideration when jobs are being dispatched.

Commands:

- ◆ `lsinfo -m`—View a list of currently running models
- ◆ `lsinfo -M`—View all models defined in `lsf.shared`

Configuration:

- ◆ Defined in `lsf.shared`
- ◆ Mapped to hosts in `lsf.cluster.cluster_name`

Users and administrators

LSF user	A user account that has permission to submit jobs to the LSF cluster.
LSF administrator	In general, you must be an LSF administrator to perform operations that will affect other LSF users. Each cluster has one primary LSF administrator, specified during LSF installation. You can also configure additional administrators at the cluster level and at the queue level.
Primary LSF administrator	The first cluster administrator specified during installation and first administrator listed in <code>lsf.cluster.cluster_name</code> . The primary LSF administrator account owns the configuration and log files. The primary LSF administrator has permission to perform clusterwide operations, change configuration files, reconfigure the cluster, and control jobs submitted by all users.
Cluster administrator	<p>May be specified during LSF installation or configured after installation. Cluster administrators can perform administrative operations on all jobs and queues in the cluster. Cluster administrators have the same cluster-wide operational privileges as the primary LSF administrator except that they do not necessarily have permission to change LSF configuration files.</p> <p>For example, a cluster administrator can create an LSF host group, submit a job to any queue, or terminate another user's job.</p>
Queue administrator	An LSF administrator user account that has administrative permissions limited to a specified queue. For example, an LSF queue administrator can perform administrative operations on the specified queue, or on jobs running in the specified queue, but cannot change LSF configuration or operate on LSF daemons.

Resources

Resource usage	<p>The LSF system uses built-in and configured resources to track resource availability and usage. Jobs are scheduled according to the resources available on individual hosts.</p> <p>Jobs submitted through the LSF system will have the resources they use monitored while they are running. This information is used to enforce resource limits and load thresholds as well as fairshare scheduling.</p> <p>LSF collects information such as:</p> <ul style="list-style-type: none">◆ Total CPU time consumed by all processes in the job◆ Total resident memory usage in KB of all currently running processes in a job◆ Total virtual memory usage in KB of all currently running processes in a job◆ Currently active process group ID in a job◆ Currently active processes in a job <p>On UNIX, job-level resource usage is collected through PIM.</p> <p>Commands:</p> <ul style="list-style-type: none">◆ <code>lsinfo</code>—View the resources available in your cluster◆ <code>bjobs -l</code>—View current resource usage of a job
----------------	---

Configuration:

- ◆ `SBD_SLEEP_TIME` in `lsb.params` — Configures how often resource usage information is sampled by PIM, collected by `sbatchd`, and sent to `mbatchd`

Load indices

Load indices measure the availability of dynamic, non-shared resources on hosts in the cluster. Load indices built into the LIM are updated at fixed time intervals.

Commands:

- ◆ `lsload -l` — View all load indices
- ◆ `bhosts -l` — View load levels on a host

External load indices

Defined and configured by the LSF administrator and collected by an External Load Information Manager (ELIM) program. The ELIM also updates LIM when new values are received.

Commands:

- ◆ `lsinfo` — View external load indices

Static resources

Built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

Load thresholds

Two types of load thresholds can be configured by your LSF administrator to schedule jobs in queues. Each load threshold specifies a load index value:

- ◆ `loadSched` determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job will not be started on the host. This threshold is also used as the condition for resuming suspended jobs.
- ◆ `loadStop` determines when running jobs should be suspended.

To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (`>`) or less than (`<`), depending on the load index.

Commands:

- ◆ `bhosts -l` — View suspending conditions for hosts
- ◆ `bqueues -l` — View suspending conditions for queues
- ◆ `bjobs -l` — View suspending conditions for a particular job and the scheduling thresholds that control when a job is resumed

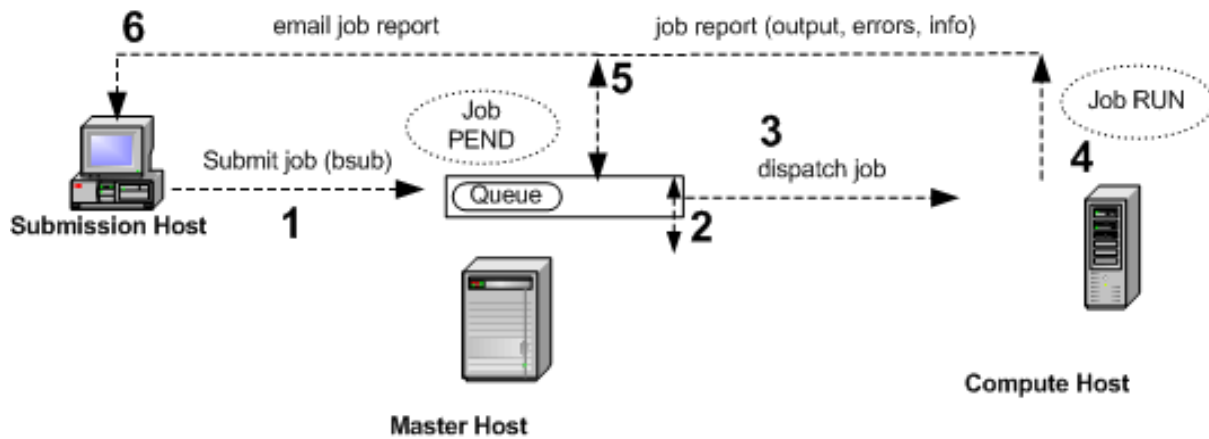
Configuration:

- ◆ `lsb.hosts` — Configure thresholds for hosts
- ◆ `lsb.queues` — Configure thresholds for queues

Cluster Concepts

- Runtime resource usage limits** Limit the use of resources while a job is running. Jobs that consume more than the specified amount of a resource are signalled or have their priority lowered.
- Configuration:
- ◆ `lsb.queues` — Configure resource usage limits for queues
- Hard and soft limits** Resource limits specified at the queue level are hard limits while those specified with job submission are soft limits. See `setrlimit(2)` man page for concepts of hard and soft limits.
- Resource allocation limits** Restrict the amount of a given resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to. If all of the resource has been consumed, no more jobs can be started until some of the resource is released.
- Configuration:
- ◆ `lsb.resources` — Configure queue-level resource allocation limits for hosts, users, queues, and projects
- Resource requirements (bsub -R)** Restrict which hosts the job can run on. Hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it collects the load index values of all the candidate hosts and compares them to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.
- Commands:
- ◆ `bsub -R` — Specify resource requirement string for a job
- Configuration:
- ◆ `lsb.queues` — Configure resource requirements for queues

Job Life Cycle



1 Submit a job

You submit a job from an LSF client or server with the `bsub` command.

If you do not specify a queue when submitting the job, the job is submitted to the default queue.

Jobs are held in a queue waiting to be scheduled and have the PENDING state. The job is held in a job file in the `LSF_SHAREDIR/cluster_name/logdir/info/` directory, or in one of its subdirectories if `MAX_INFO_DIRS` is defined in `lsb.params`.

Job ID

LSF assigns each job a unique job ID when you submit the job.

Job name

You can also assign a name to the job with the `-J` option of `bsub`. Unlike the job ID, the job name is not necessarily unique.

2 Schedule job

1 `mbatchd` looks at jobs in the queue and sends the jobs for scheduling to `mbschd` at a preset time interval (defined by the parameter `JOB_SCHEDULING_INTERVAL` in `lsb.params`).

2 `mbschd` evaluates jobs and makes scheduling decisions based on:

- ❖ Job priority
- ❖ Scheduling policies
- ❖ Available resources

3 `mbschd` selects the best hosts where the job can run and sends its decisions back to `mbatchd`.

Resource information is collected at preset time intervals by the master LIM from LIMs on server hosts. The master LIM communicates this information to `mbatchd`, which in turn communicates it to `mbschd` to support scheduling decisions.

3 Dispatch job

As soon as `mbatchd` receives scheduling decisions, it immediately dispatches the jobs to hosts.

4 Run job

`sbatchd` handles job execution. It:

- 1 Receives the request from `mbatchd`
- 2 Creates a child `sbatchd` for the job
- 3 Creates the execution environment
- 4 Starts the job using `res`

The execution environment is copied from the submission host to the execution host and includes the following:

- ❖ Environment variables needed by the job
- ❖ Working directory where the job begins running
- ❖ Other system-dependent environment settings, for example:
 - ❖ On UNIX, resource limits and `umask`
 - ❖ On Windows, desktop and Windows root directory

The job runs under the user account that submitted the job and has the status `RUN`.

5 Return output

When a job is completed, it is assigned the `DONE` status if the job was completed without any problems. The job is assigned the `EXIT` status if errors prevented the job from completing.

`sbatchd` communicates job information including errors and output to `mbatchd`.

6 Send email to client

`mbatchd` returns the job output, job error, and job information to the submission host through email. Use the `-o` and `-e` options of `bsub` to send job output and errors to a file.

Job report

A job report is sent by email to the LSF client and includes:

- ◆ Job information such as:
 - ❖ CPU use
 - ❖ Memory use
 - ❖ Name of the account that submitted the job
- ◆ Job output
- ◆ Errors

How the System Works

LSF can be configured in different ways that affect the scheduling of jobs. By default, this is how LSF handles a new job:

- 1 Receive the job. Create a job file. Return the job ID to the user.
- 2 Schedule the job and select the best available host.
- 3 Dispatch the job to a selected host.
- 4 Set the environment on the host.
- 5 Start the job.

Contents

- ◆ [Job Submission](#) on page 27
- ◆ [Job Scheduling and Dispatch](#) on page 29
- ◆ [Host Selection](#) on page 31
- ◆ [Job Execution Environment](#) on page 32
- ◆ [Fault Tolerance](#) on page 33

Job Submission

The life cycle of a job starts when you submit the job to LSF. On the command line, `bsub` is used to submit jobs, and you can specify many options to `bsub` to modify the default behavior, including the use of a JSDL file. Jobs must be submitted to a queue.

Queues

Queues represent a set of pending jobs, lined up in a defined order and waiting for their opportunity to use resources. Queues implement different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Queues do not correspond to individual hosts; each queue can use all server hosts in the cluster, or a configured subset of the server hosts.

A queue is a network-wide holding place for jobs. Jobs enter the queue via the `bsub` command. LSF can be configured to have one or more default queues. Jobs that are not submitted to a specific queue will be assigned to the first default queue that accepts them. Queues have the following attributes associated with them:

- ◆ Priority, where a larger integer is a higher priority
- ◆ Name, which uniquely identifies the queue
- ◆ Queue limits, that restrict hosts, number of jobs, users, groups, processors, etc.
- ◆ Standard UNIX limits: memory, swap, process, CPU, etc.
- ◆ Scheduling policies: FCFS, fairshare, preemptive, exclusive
- ◆ Administrators
- ◆ Run conditions
- ◆ Load-sharing threshold conditions, which apply load sharing to the queue
- ◆ UNIX `nice(1)` value, which sets the UNIX scheduler priority

Example queue

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
STACKLIMIT= 2048
DESCRIPTION = For normal low priority jobs, running only if hosts
are lightly loaded.
QJOB_LIMIT = 60      # job limit of the queue
PJOB_LIMIT = 2      # job limit per processor
ut = 0.2
io = 50/240
USERS = all
HOSTS = all
NICE = 20
End Queue
```

Queue priority

Defines the order in which queues are searched to determine which job will be processed. Queues are assigned a priority by the LSF administrator, where a higher number has a higher priority. Queues are serviced by LSF in order of priority from the highest to the lowest. If multiple queues have the same priority, LSF schedules all the jobs from these queues in first-come, first-served order.

Automatic queue selection

Typically, a cluster has multiple queues. When you submit a job to LSF you might define which queue the job will enter. If you submit a job without specifying a queue name, LSF considers the requirements of the job and automatically chooses a suitable queue from a list of candidate default queues. If you did not define any candidate default queues, LSF will create a new queue using all the default settings, and submit the job to that queue.

Viewing default queues

Use `bparams` to display default queues:

```
bparams
Default Queues: normal
...
```

The user can override this list by defining the environment variable `LSB_DEFAULTQUEUE`.

How automatic queue selection works

LSF selects a suitable queue according to:

- ◆ User access restriction—Queues that do not allow this user to submit jobs are not considered.
- ◆ Host restriction—If the job explicitly specifies a list of hosts on which the job can be run, then the selected queue must be configured to send jobs to all hosts in the list.
- ◆ Queue status—Closed queues are not considered.
- ◆ Exclusive execution restriction—If the job requires exclusive execution, then queues that are not configured to accept exclusive jobs are not considered.
- ◆ Job's requested resources—These must be within the resource allocation limits of the selected queue.

If multiple queues satisfy the above requirements, then the first queue listed in the candidate queues (as defined by the `DEFAULT_QUEUE` parameter or the `LSB_DEFAULTQUEUE` environment variable) that satisfies the requirements is selected.

Job files

When a batch job is submitted to a queue, LSF Batch holds it in a job file until conditions are right for it to be executed. Then the job file is used to execute the job.

UNIX

The job file is a Bourne shell script run at execution time.

Windows

The job file is a batch file processed at execution time.

Job Scheduling and Dispatch

Submitted jobs sit in queues until they are scheduled and dispatched to a host for execution. When a job is submitted to LSF, many factors control when and where the job starts to run:

- ◆ Active time window of the queue or hosts
- ◆ Resource requirements of the job
- ◆ Availability of eligible hosts
- ◆ Various job slot limits
- ◆ Job dependency conditions
- ◆ Fairshare constraints
- ◆ Load conditions

Scheduling policies

First-Come, First-Served (FCFS) scheduling

By default, jobs in a queue are dispatched in first-come, first-served (FCFS) order. This means that jobs are dispatched according to their order in the queue. Since jobs are ordered according to job priority, this does not necessarily mean that jobs will be dispatched in the order of submission. The order of jobs in the queue can also be modified by the user or administrator.

Service level agreement (SLA) scheduling

An SLA in LSF is a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users. The SLA scheduling policy defines how many jobs should be run from each SLA to meet the configured goals.

Fairshare scheduling and other policies

If a fairshare scheduling policy has been specified for the queue or if host partitions have been configured, jobs are dispatched in accordance with these policies instead. To solve diverse problems, LSF allows multiple scheduling policies in the same cluster. LSF has several queue scheduling policies such as exclusive, preemptive, fairshare, and hierarchical fairshare.

Scheduling and dispatch

Jobs are scheduled at regular intervals (5 seconds by default, configured by the parameter `JOB_SCHEDULING_INTERVAL` in `lsb.params`). Once jobs are scheduled, they can be immediately dispatched to hosts.

To prevent overloading any host, LSF waits a short time between dispatching jobs to the same host. The delay is configured by the `JOB_ACCEPT_INTERVAL` parameter in `lsb.params` or `lsb.queues`; the default is 60 seconds. If `JOB_ACCEPT_INTERVAL` is set to zero, more than one job can be started on a host at a time.

For large clusters, define `LSF_SERVER_HOSTS` in `lsf.conf` to decrease the load on the master LIM.

Some operating systems, such as Linux and AIX, let you increase the number of file descriptors that can be allocated to the master host. You do not need to limit the number of file descriptors to 1024 if you want fast job dispatching. To take advantage of the greater number of file descriptors, you must set the parameter `LSB_MAX_JOB_DISPATCH_PER_SESSION` in `lsf.conf` to a value greater than 300 and less than or equal to one-half the value of `MAX_SBD_CONNS` defined in `lsb.params`. `LSB_MAX_JOB_DISPATCH_PER_SESSION` defines the maximum number of jobs that `mbatchd` can dispatch during one job scheduling session. You must restart `mbatchd` and `sbatchd` when you change the value of this parameter for the change to take effect.

Dispatch order

Jobs are not necessarily dispatched in order of submission.

Each queue has a priority number set by an LSF Administrator when the queue is defined. LSF tries to start jobs from the highest priority queue first.

By default, LSF considers jobs for dispatch in the following order:

- ◆ For each queue, from highest to lowest priority. If multiple queues have the same priority, LSF schedules all the jobs from these queues in first-come, first-served order.
- ◆ For each job in the queue, according to FCFS order
- ◆ If any host is eligible to run this job, start the job on the best eligible host, and mark that host ineligible to start any other job until `JOB_ACCEPT_INTERVAL` has passed

Jobs can be dispatched out of turn if pre-execution conditions are not met, specific hosts or resources are busy or unavailable, or a user has reached the user job slot limit.

Viewing job order in queue

Use `bjobs` to see the order in which jobs in a queue will actually be dispatched for the FCFS policy.

Changing job order in queue (`btop` and `bbot`)

Use the `btop` and `bbot` commands to change the job order in the queue.

See [Changing Job Order Within Queues](#) on page 109 for more information.

Host Selection

Each time LSF attempts to dispatch a job, it checks to see which hosts are eligible to run the job. A number of conditions determine whether a host is eligible:

- ◆ Host dispatch windows
- ◆ Resource requirements of the job
- ◆ Resource requirements of the queue
- ◆ Host list of the queue
- ◆ Host load levels
- ◆ Job slot limits of the host.

A host is only eligible to run a job if all the conditions are met. If a job is queued and there is an eligible host for that job, the job is placed on that host. If more than one host is eligible, the job is started on the best host based on both the job and the queue resource requirements.

Host load levels

A host is available if the values of the load indices (such as `r1m`, `pg`, `mem`) of the host are within the configured scheduling thresholds. There are two sets of scheduling thresholds: host and queue. If any load index on the host exceeds the corresponding host threshold or queue threshold, the host is not eligible to run any job.

Viewing host load levels

- ◆ Use the `bhosts -l` command to display the host thresholds.
- ◆ Use the `bqueues -l` command to display the queue thresholds.

Eligible hosts

When LSF tries to place a job, it obtains current load information for all hosts.

The load levels on each host are compared to the scheduling thresholds configured for that host in the `Host` section of `lsb.hosts`, as well as the per-queue scheduling thresholds configured in `lsb.queues`.

If any load index exceeds either its per-queue or its per-host scheduling threshold, no new job is started on that host.

Viewing eligible hosts

The `bjobs -lp` command displays the names of hosts that cannot accept a job at the moment together with the reasons the job cannot be accepted.

Resource requirements

Resource requirements at the queue level can also be used to specify scheduling conditions (for example, `r1m<0.4 && pg<3`).

A higher priority or earlier batch job is only bypassed if no hosts are available that meet the requirements of that job.

If a host is available but is not eligible to run a particular job, LSF looks for a later job to start on that host. LSF starts the first job found for which that host is eligible.

Job Execution Environment

When LSF runs your jobs, it tries to make it as transparent to the user as possible. By default, the execution environment is maintained to be as close to the submission environment as possible. LSF will copy the environment from the submission host to the execution host. The execution environment includes the following:

- ◆ Environment variables needed by the job
- ◆ Working directory where the job begins running
- ◆ Other system-dependent environment settings; for example, resource usage limits and `umask`:

Since a network can be heterogeneous, it is often impossible or undesirable to reproduce the submission host's execution environment on the execution host. For example, if home directory is not shared between submission and execution host, LSF runs the job in the `/tmp` on the execution host. If the `DISPLAY` environment variable is something like `Unix:0.0`, or `:0.0`, then it must be processed before using on the execution host. These are automatically handled by LSF.

To change the default execution environment, use:

- ◆ A job starter
- ◆ `bsub -L`

For resource control, LSF also changes some of the execution environment of jobs. These include nice values, resource usage limits, or any other environment by configuring a job starter.

Shared user directories

LSF works best when user home directories are shared across all hosts in the cluster. To provide transparent remote execution, you should share user home directories on all LSF hosts.

To provide transparent remote execution, LSF commands determine the user's current working directory and use that directory on the remote host.

For example, if the command `cc file.c` is executed remotely, `cc` only finds the correct `file.c` if the remote command runs in the same directory.

LSF automatically creates an `.lsbatch` subdirectory in the user's home directory on the execution host. This directory is used to store temporary input and output files for jobs.

Executables and the PATH environment variable

Search paths for executables (the `PATH` environment variable) are passed to the remote execution host unchanged. In mixed clusters, LSF works best when the user binary directories (for example, `/usr/bin`, `/usr/local/bin`) have the same path names on different host types. This makes the `PATH` variable valid on all hosts.

LSF configuration files are normally stored in a shared directory. This makes administration easier. There is little performance penalty for this, because the configuration files are not frequently read.

Fault Tolerance

LSF is designed to continue operating even if some of the hosts in the cluster are unavailable. One host in the cluster acts as the master, but if the master host becomes unavailable another host takes over. LSF is available as long as there is one available host in the cluster.

LSF can tolerate the failure of any host or group of hosts in the cluster. When a host crashes, all jobs running on that host are lost. No other pending or running jobs are affected. Important jobs can be submitted to LSF with an option to automatically restart if the job is lost because of a host failure.

Dynamic master host

The LSF master host is chosen dynamically. If the current master host becomes unavailable, another host takes over automatically. The failover master host is selected from the list defined in `LSF_MASTER_LIST` in `lsf.conf` (specified in `install.config` at installation). The first available host in the list acts as the master. LSF might be unavailable for a few minutes while hosts are waiting to be contacted by the new master.

Running jobs are managed by `sbatchd` on each server host. When the new `mbatchd` starts, it polls the `sbatchd` on each host and finds the current status of its jobs. If `sbatchd` fails but the host is still running, jobs running on the host are not lost. When `sbatchd` is restarted it regains control of all jobs running on the host.

Network failure

If the cluster is partitioned by a network failure, a master LIM takes over on each side of the partition. Interactive load-sharing remains available, as long as each host still has access to the LSF executables.

Event log file (`lsb.events`)

Fault tolerance in LSF depends on the event log file, `lsb.events`, which is kept on the primary file server. Every event in the system is logged in this file, including all job submissions and job and host status changes. If the master host becomes unavailable, a new master is chosen by `lim`. `sbatchd` on the new master starts a new `mbatchd`. The new `mbatchd` reads the `lsb.events` file to recover the state of the system.

For sites not wanting to rely solely on a central file server for recovery information, LSF can be configured to maintain a duplicate event log by keeping a replica of `lsb.events`. The replica is stored on the file server, and used if the primary copy is unavailable. When using LSF's duplicate event log function, the primary event log is stored on the first master host, and re-synchronized with the replicated copy when the host recovers.

Partitioned network

If the network is partitioned, only one of the partitions can access `lsb.events`, so batch services are only available on one side of the partition. A lock file is used to make sure that only one `mbatchd` is running in the cluster.

Host failure

If an LSF server host fails, jobs running on that host are lost. No other jobs are affected. Jobs can be submitted as *rerunnable*, so that they automatically run again from the beginning or as *checkpointable*, so that they start again from a checkpoint on another host if they are lost because of a host failure.

If all of the hosts in a cluster go down, all running jobs are lost. When a host comes back up and takes over as master, it reads the `lsb.events` file to get the state of all batch jobs. Jobs that were running when the systems went down are assumed to have exited, and email is sent to the submitting user. Pending jobs remain in their queues, and are scheduled as hosts become available.

Job exception handling

You can configure hosts and queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

See [Handling Host-level Job Exceptions](#) on page 88 and [Handling Job Exceptions in Queues](#) on page 100 for more information about job-level exception management.

Managing Your Cluster

- ◆ [Working with Your Cluster](#) on page 37
- ◆ [Working with Hosts](#) on page 53
- ◆ [Working with Queues](#) on page 91
- ◆ [Managing Jobs](#) on page 103
- ◆ [Managing Users and User Groups](#) on page 129
- ◆ [Platform LSF Licensing](#) on page 135
- ◆ [Managing LSF on Platform EGO](#) on page 169
- ◆ [Cluster Version Management and Patching on UNIX and Linux](#) on page 211

Working with Your Cluster

Contents

- ◆ [Viewing cluster information](#) on page 38
- ◆ [Example directory structures](#) on page 43
- ◆ [Cluster administrators](#) on page 45
- ◆ [Controlling daemons](#) on page 46
- ◆ [Controlling mbatchd](#) on page 48
- ◆ [Reconfiguring your cluster](#) on page 51

Viewing cluster information

LSF provides commands for users to access information about the cluster. Cluster information includes the cluster master host, cluster name, cluster resource definitions, cluster administrator, and so on.

To view the ...	Run ...
Version of LSF	<code>lsid</code>
Cluster name	<code>lsid</code>
Current master host	<code>lsid</code>
Cluster administrators	<code>lscusters</code>
Configuration parameters	<code>bparams</code>

View LSF version, cluster name, and current master host

- 1 Run `lsid` to display the version of LSF, the name of your cluster, and the current master host:

```
lsid
Platform LSF 7 Update 2 Feb 07 2007
Copyright 1992-2008 Platform Computing Corporation

My cluster name is cluster1
My master name is hostA
```

View cluster administrators

- 1 Run `lscusters` to find out who your cluster administrator is and see a summary of your cluster:

```
lscusters
CLUSTER_NAME STATUS MASTER_HOST ADMIN HOSTS SERVERS
cluster1 ok hostA lsfadmin 6 6
```

If you are using the LSF MultiCluster product, you will see one line for each of the clusters that your local cluster is connected to in the output of `lscusters`.

View configuration parameters

- 1 Run `bparams` to display the generic configuration parameters of LSF. These include default queues, job dispatch interval, job checking interval, and job accepting interval.

```
bparams
Default Queues: normal idle
Job Dispatch Interval: 20 seconds
Job Checking Interval: 15 seconds
Job Accepting Interval: 20 seconds
```

- 2 Run `bparams -l` to display the information in long format, which gives a brief description of each parameter and the name of the parameter as it appears in `lsb.params`.

```
bparams -l
```

```
System default queues for automatic queue selection:
```

```
  DEFAULT_QUEUE = normal idle
```

```
The interval for dispatching jobs by master batch daemon:
```

```
  MBD_SLEEP_TIME = 20 (seconds)
```

```
The interval for checking jobs by slave batch daemon:
```

```
  SBD_SLEEP_TIME = 15 (seconds)
```

```
The interval for a host to accept two batch jobs subsequently:
```

```
  JOB_ACCEPT_INTERVAL = 1 (* MBD_SLEEP_TIME)
```

```
The idle time of a host for resuming pg suspended jobs:
```

```
  PG_SUSP_IT = 180 (seconds)
```

```
The amount of time during which finished jobs are kept in core:
```

```
  CLEAN_PERIOD = 3600 (seconds)
```

```
The maximum number of finished jobs that are logged in current event file:
```

```
  MAX_JOB_NUM = 2000
```

```
The maximum number of retries for reaching a slave batch daemon:
```

```
  MAX_SBD_FAIL = 3
```

```
The number of hours of resource consumption history:
```

```
  HIST_HOURS = 5
```

```
The default project assigned to jobs.
```

```
  DEFAULT_PROJECT = default
```

```
Sync up host status with master LIM is enabled:
```

```
  LSB_SYNC_HOST_STAT_LIM = Y
```

```
MBD child query processes will only run on the following CPUs:
```

```
  MBD_QUERY_CPUS=1 2 3
```

- 3 Run `bparams -a` to display all configuration parameters and their values in `lsb.params`.

For example:

```
bparams -a
```

```
lsb.params configuration at Fri Jun 8 10:27:52 CST 2007
```

```
  MBD_SLEEP_TIME = 20
```

```
  SBD_SLEEP_TIME = 15
```

```
  JOB_ACCEPT_INTERVAL = 1
```

```
  SUB_TRY_INTERVAL = 60
```

```
  LSB_SYNC_HOST_STAT_LIM = N
```

```
  MAX_JOBINFO_QUERY_PERIOD = 2147483647
```

```
  PEND_REASON_UPDATE_INTERVAL = 30
```

Viewing daemon parameter configuration

- 1 Display all configuration settings for running LSF daemons.
 - ❖ Use `lsadmin showconf` to display all configured parameters and their values in `lsf.conf` or `ego.conf` for LIM.
 - ❖ Use `badmin showconf` to display all configured parameters and their values in `lsf.conf` or `ego.conf` for `mbatchd` and `sbatchd`.

In a MultiCluster environment, `lsadmin showconf` and `badmin showconf` only display the parameters of daemons on the local cluster.

Running `lsadmin showconf` and `badmin showconf` from a master candidate host will reach all server hosts in the cluster. Running `lsadmin showconf` and `badmin showconf` from a slave-only host may not be able to reach other slave-only hosts.

You cannot run `lsadmin showconf` and `badmin showconf` from client hosts. `lsadmin` shows only server host configuration, not client host configuration.

`lsadmin showconf` and `badmin showconf` only displays the values used by LSF.

`lsadmin showconf` and `badmin showconf` display `EGO_MASTER_LIST` from wherever it is defined. You can define either `LSF_MASTER_LIST` in `lsf.conf` or `EGO_MASTER_LIST` in `ego.conf`. LIM reads `lsf.conf` first, and `ego.conf` if EGO is enabled in the LSF cluster. LIM only takes the value of `LSF_MASTER_LIST` if `EGO_MASTER_LIST` is not defined at all in `lsf.conf`.

For example, if EGO is enabled in the LSF cluster, and you define `LSF_MASTER_LIST` in `lsf.conf`, and `EGO_MASTER_LIST` in `ego.conf`, `lsadmin showconf` and `badmin showconf` display the value of `EGO_MASTER_LIST` in `ego.conf`.

If EGO is disabled, `ego.conf` not loaded, so whatever is defined in `lsf.conf` is displayed.

- 2 Display `mbatchd` and root `sbatchd` configuration.
 - ❖ Use `badmin showconf mbd` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to `mbatchd`.
 - ❖ Use `badmin showconf sbd` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to root `sbatchd`.
- 3 Display LIM configuration.

Use `lsadmin showconf lim` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to root LIM.

By default, `lsadmin` displays the local LIM parameters. You can specify the host to display the LIM parameters.

Examples

- ◆ Show `mbatchd` configuration:


```
badmin showconf mbd
MBD configuration at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
```



```
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

- ◆ **Show sbatchd configuration on a specific host:**

```
badmin showconf sbd hosta
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

- ◆ **Show sbatchd configuration for all hosts:**

```
badmin showconf sbd all
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

```
SBD configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

- ◆ **Show lim configuration:**

```
lsadmin showconf lim
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

- ◆ **Show lim configuration for a specific host:**

```
lsadmin showconf lim hosta
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
```

Viewing cluster information

```
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

- ◆ **Show lim configuration for all hosts:**

```
lsadmin showconf lim all
```

```
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
```

```
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

```
LIM configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2007
```

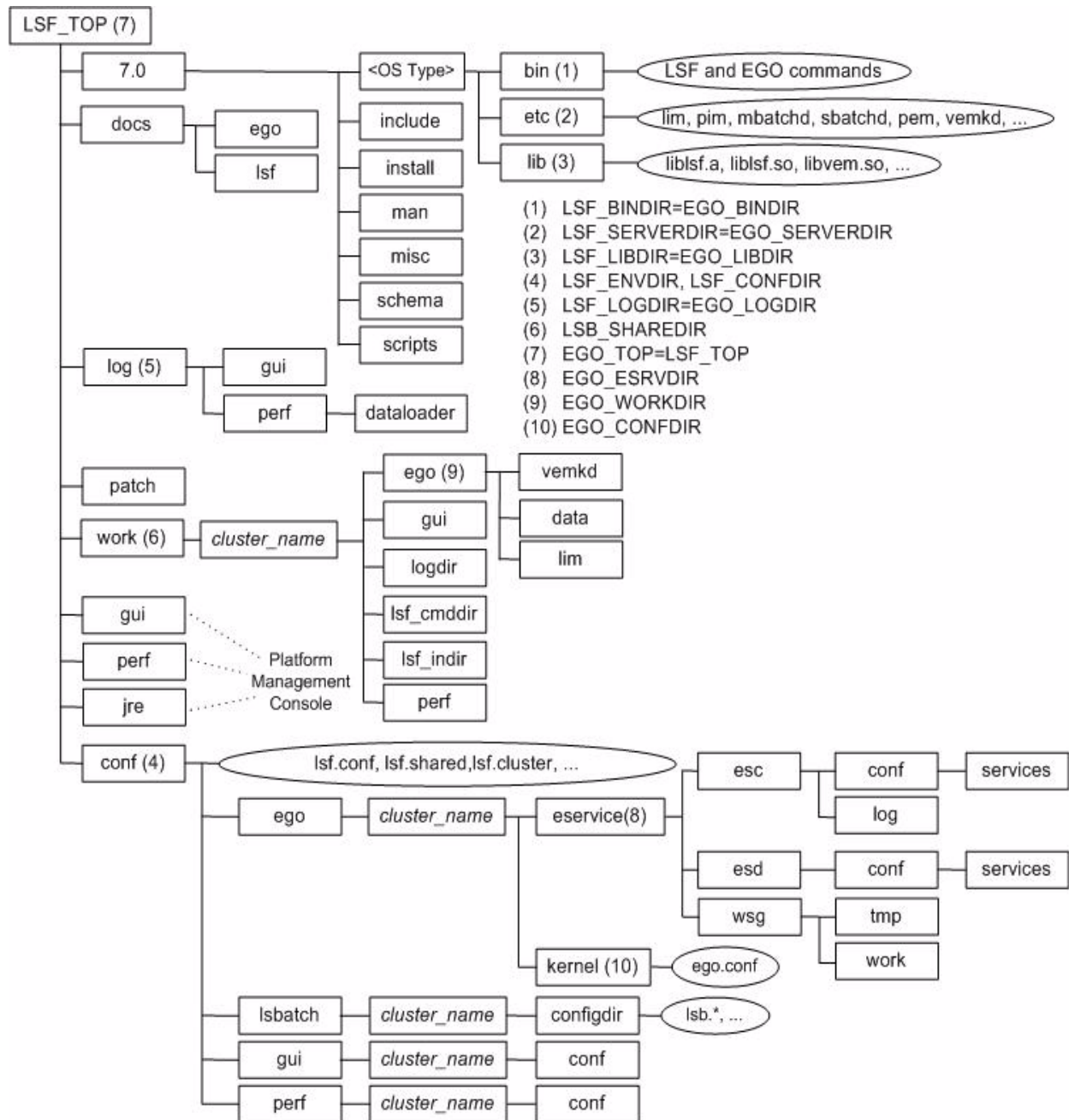
```
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

Example directory structures

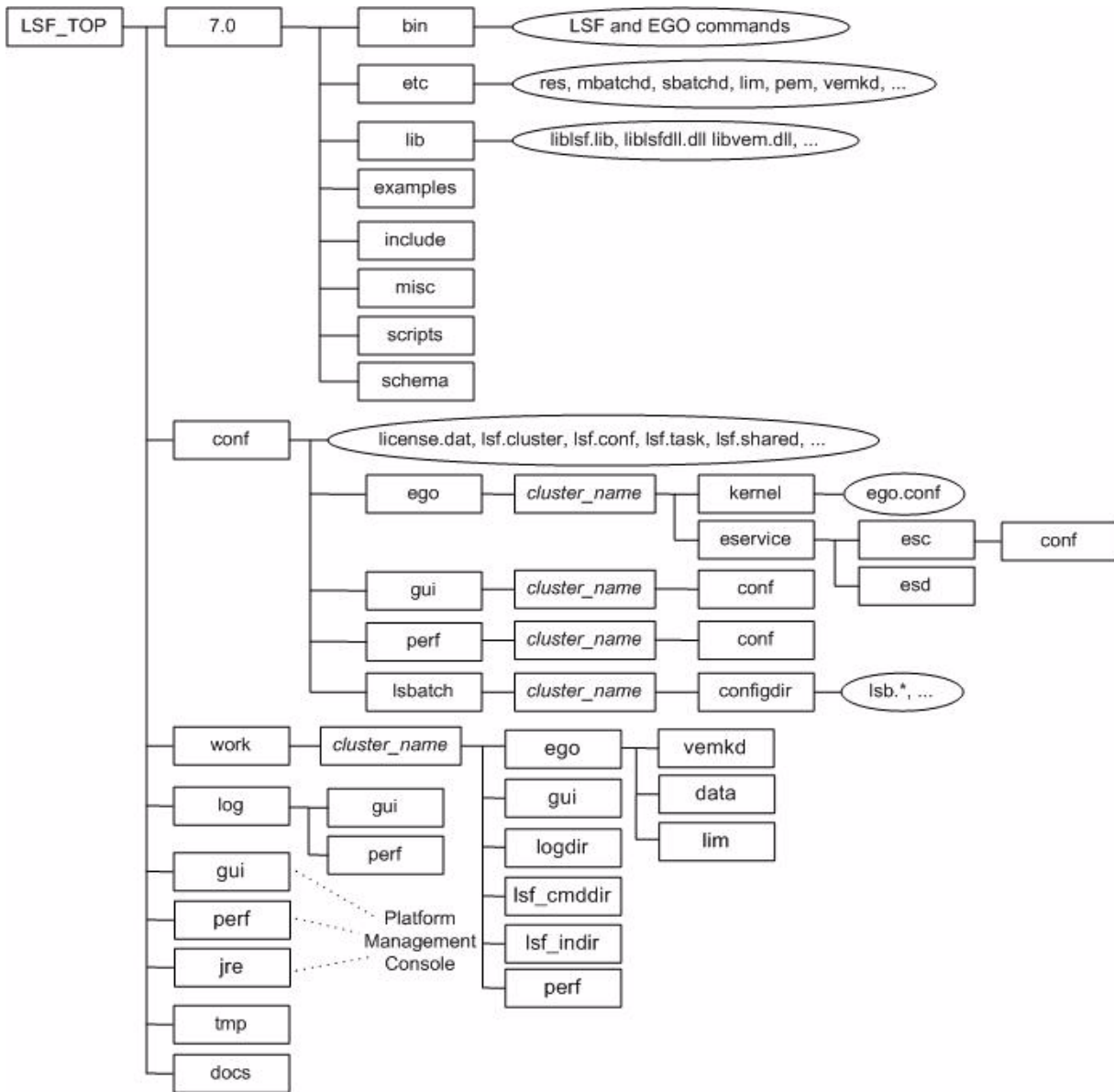
UNIX and Linux

The following figures show typical directory structures for a new UNIX or Linux installation with `lsfinstall`. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Microsoft Windows

The following diagram shows an example directory structure for a Windows installation.



Cluster administrators

Primary cluster administrator

Required. The first cluster administrator, specified during installation. The primary LSF administrator account owns the configuration and log files. The primary LSF administrator has permission to perform clusterwide operations, change configuration files, reconfigure the cluster, and control jobs submitted by all users.

Other cluster administrators

Optional. May be configured during or after installation.

Cluster administrators can perform administrative operations on all jobs and queues in the cluster. Cluster administrators have the same cluster-wide operational privileges as the primary LSF administrator except that they do not have permission to change LSF configuration files.

Add cluster administrators

- 1 In the `ClusterAdmins` section of `lsf.cluster.cluster_name`, specify the list of cluster administrators following `ADMINISTRATORS`, separated by spaces.

You can specify user names and group names.

The first administrator in the list is the primary LSF administrator. All others are cluster administrators.

For example:

```
Begin ClusterAdmins
ADMINISTRATORS = lsfadmin admin1 admin2
End ClusterAdmins
```

- 2 Save your changes.
- 3 Run `lsadmin reconfig` to reconfigure LIM.
- 4 Run `badmin mbdrestart` to restart `mbatchd`.

Controlling daemons

Permissions required

To control all daemons in the cluster, you must

- ◆ Be logged on as root or as a user listed in the `/etc/lsf.sudoers` file. See the *Platform LSF Configuration Reference* for configuration details of `lsf.sudoers`.
- ◆ Be able to run the `rsh` or `ssh` commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring the `rsh` and `ssh` commands. The shell command specified by `LSF_RSH` in `lsf.conf` is used before `rsh` is tried.

Daemon commands

The following is an overview of commands you use to control LSF daemons.

Daemon	Action	Command	Permissions
All in cluster	Start	<code>lsfstartup</code>	Must be root or a user listed in <code>lsf.sudoers</code> for all these commands
	Shut down	<code>lsfshutdown</code>	
sbatchd	Start	<code>badadmin hstartup [host_name ... all]</code>	Must be root or a user listed in <code>lsf.sudoers</code> for the startup command
	Restart	<code>badadmin hrestart [host_name ... all]</code>	Must be root or the LSF administrator for other commands
	Shut down	<code>badadmin hshutdown [host_name ... all]</code>	
mbatchd mbschd	Restart	<code>badadmin mbdrestart</code>	Must be root or the LSF administrator for these commands
	Shut down	<ol style="list-style-type: none"> 1 <code>badadmin hshutdown</code> 2 <code>badadmin mbdrestart</code> 	
	Reconfigure	<code>badadmin reconfig</code>	
RES	Start	<code>lsadmin resstartup [host_name ... all]</code>	Must be root or a user listed in <code>lsf.sudoers</code> for the startup command
	Shut down	<code>lsadmin resshutdown [host_name ... all]</code>	Must be the LSF administrator for other commands
	Restart	<code>lsadmin resrestart [host_name ... all]</code>	
LIM	Start	<code>lsadmin limstartup [host_name ... all]</code>	Must be root or a user listed in <code>lsf.sudoers</code> for the startup command
	Shut down	<code>lsadmin limshutdown [host_name ... all]</code>	Must be the LSF administrator for other commands
	Restart	<code>lsadmin limrestart [host_name ... all]</code>	
	Restartall in cluster	<code>lsadmin reconfig</code>	

sbatchd

Restarting `sbatchd` on a host does not affect jobs that are running on that host.

If `sbatchd` is shut down, the host is not available to run new jobs. Existing jobs running on that host continue, but the results are not sent to the user until `sbatchd` is restarted.

LIM and RES

Jobs running on the host are not affected by restarting the daemons.

If a daemon is not responding to network connections, `lsadmin` displays an error message with the host name. In this case you must kill and restart the daemon manually.

If the LIM and the other daemons on the current master host shut down, another host automatically takes over as master.

If the RES is shut down while remote interactive tasks are running on the host, the running tasks continue but no new tasks are accepted.

Controlling mbatchd

You use the `badadmin` command to control `mbatchd`.

Reconfigure mbatchd

If you add a host to a host group, a host to a queue, or change resource configuration in the Hosts section of `lsf.cluster.cluster_name`, the change is not recognized by jobs that were submitted before you reconfigured. If you want the new host to be recognized, you must restart `mbatchd`.

- 1 Run `badadmin reconfig`.

When you reconfigure the cluster, `mbatchd` is not restarted. Only configuration files are reloaded.

Restart mbatchd

- 1 Run `badadmin mbdrestart`.

LSF checks configuration files for errors and prints the results to `stderr`. If no errors are found, the following occurs:

- ◆ Configuration files are reloaded
 - ◆ `mbatchd` is restarted
 - ◆ Events in `lsb.events` are reread and replayed to recover the running state of the last `mbatchd`
-

TIP: Whenever `mbatchd` is restarted, it is unavailable to service requests. In large clusters where there are many events in `lsb.events`, restarting `mbatchd` can take some time. To avoid replaying events in `lsb.events`, use the command `badadmin reconfig`.

Log a comment when restarting mbatchd

- 1 Use the `-C` option of `badadmin mbdrestart` to log an administrator comment in `lsb.events`.

For example:

```
badadmin mbdrestart -C "Configuration change"
```

The comment text `Configuration change` is recorded in `lsb.events`.

- 2 Run `badadmin hist` or `badadmin mbdhist` to display administrator comments for `mbatchd` restart.

Shut down mbatchd

- 1 Run `badmin hshutdown` to shut down `sbatchd` on the master host.

For example:

```
badmin hshutdown hostD
Shut down slave batch daemon on <hostD> .... done
```

- 2 Run `badmin mbdrestart`:

```
badmin mbdrestart
Checking configuration files ...
No errors found.
```

This causes `mbatchd` and `mbschd` to exit. `mbatchd` cannot be restarted, because `sbatchd` is shut down. All LSF services are temporarily unavailable, but existing jobs are not affected. When `mbatchd` is later started by `sbatchd`, its previous status is restored from the event log file and job scheduling continues.

Customize batch command messages

LSF displays error messages when a batch command cannot communicate with `mbatchd`. Users see these messages when the batch command retries the connection to `mbatchd`.

You can customize three of these messages to provide LSF users with more detailed information and instructions.

- 1 In the file `lsf.conf`, identify the parameter for the message that you want to customize.

The following lists the parameters you can use to customize messages when a batch command does not receive a response from `mbatchd`.

Reason for no response from <code>mbatchd</code>	Default message	Parameter used to customize the message
<code>mbatchd</code> is too busy to accept new connections or respond to client requests	LSF is processing your request. Please wait...	<code>LSB_MBD_BUSY_MSG</code>
internal system connections to <code>mbatchd</code> fail	Cannot connect to LSF. Please wait...	<code>LSB_MBD_CONNECT_FAIL_MSG</code>
<code>mbatchd</code> is down or there is no process listening at either the <code>LSB_MBD_PORT</code> or the <code>LSB_QUERY_PORT</code>	LSF is down. Please wait...	<code>LSB_MBD_DOWN_MSG</code>

- 2 Specify a message string, or specify an empty string:
 - ❖ To specify a message string, enclose the message text in quotation marks (") as shown in the following example:


```
LSB_MBD_BUSY_MSG="The mbatchd daemon is busy. Your command will retry every 5 minutes. No action required."
```
 - ❖ To specify an empty string, type quotation marks (") as shown in the following example:


```
LSB_MBD_BUSY_MSG=""
```

Whether you specify a message string or an empty string, or leave the parameter undefined, the batch command retries the connection to `mbatchd` at the intervals specified by the parameters `LSB_API_CONNTIMEOUT` and `LSB_API_RECVMTIMEOUT`.

NOTE: Before Version 7.0, LSF displayed the following message for all three message types: "batch daemon not responding...still trying." To display the previous default message, you must define each of the three message parameters and specify "batch daemon not responding...still trying" as the message string.

- 3 Save and close the `lsf.conf` file.

Reconfiguring your cluster

After changing LSF configuration files, you must tell LSF to reread the files to update the configuration. Use the following commands to reconfigure a cluster:

- ◆ `lsadmin reconfig`
- ◆ `badadmin reconfig`
- ◆ `badadmin mbdrestart`

The reconfiguration commands you use depend on which files you change in LSF. The following table is a quick reference.

After making changes to ...	Use ...	Which ...
<code>hosts</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.applications</code>	<code>badadmin reconfig</code>	reloads configuration files Pending jobs use new application profile definition. Running jobs are not affected.
<code>lsb.hosts</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.modules</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.nqsmaps</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.params</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.queues</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.resources</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.serviceclasses</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsb.users</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsf.cluster.<i>cluster_name</i></code>	<code>lsadmin reconfig</code> AND <code>badadmin mbdrestart</code>	reconfigures LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.conf</code>	<code>lsadmin reconfig</code> AND <code>badadmin mbdrestart</code>	reconfigures LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.licensescheduler</code>	<code>bladmin reconfig</code> <code>lsadmin reconfig</code> <code>badadmin mbdrestart</code>	reconfigures <code>bld</code> , reconfigures LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.shared</code>	<code>lsadmin reconfig</code> AND <code>badadmin mbdrestart</code>	reconfigures LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.sudoers</code>	<code>badadmin reconfig</code>	reloads configuration files
<code>lsf.task</code>	<code>lsadmin reconfig</code> AND <code>badadmin reconfig</code>	reconfigures LIM and reloads configuration files

Reconfigure the cluster with `lsadmin` and `badadmin`

To make a configuration change take effect, use this method to reconfigure the cluster.

- 1 Log on to the host as `root` or the LSF administrator.
- 2 Run `lsadmin reconfig` to reconfigure LIM:

```
lsadmin reconfig
```

The `lsadmin reconfig` command checks for configuration errors.

If no errors are found, you are prompted to either restart `lim` on master host candidates only, or to confirm that you want to restart `lim` on all hosts. If fatal errors are found, reconfiguration is aborted.

Reconfiguring your cluster

- 3 Run `badmin reconfig` to reconfigure `mbatchd`:

```
badmin reconfig
```

The `badmin reconfig` command checks for configuration errors.

If fatal errors are found, reconfiguration is aborted.

Reconfigure the cluster by restarting `mbatchd`

To replay and recover the running state of the cluster, use this method to reconfigure the cluster.

- 1 Run `badmin mbdrestart` to restart `mbatchd`:

```
badmin mbdrestart
```

The `badmin mbdrestart` command checks for configuration errors.

If no fatal errors are found, you are asked to confirm `mbatchd` restart. If fatal errors are found, the command exits without taking any action.

TIP: If the `lsb.events` file is large, or many jobs are running, restarting `mbatchd` can take some time. In addition, `mbatchd` is not available to service requests while it is restarted.

View configuration errors

- 1 Run `lsadmin ckconfig -v`.

- 2 Run `badmin ckconfig -v`.
-

This reports all errors to your terminal.

How reconfiguring the cluster affects licenses

If the license server goes down, LSF can continue to operate for a period of time until it attempts to renew licenses.

Reconfiguring causes LSF to renew licenses. If no license server is available, LSF does not reconfigure the system because the system would lose all its licenses and stop working.

If you have multiple license servers, reconfiguration proceeds provided LSF can contact at least one license server. In this case, LSF still loses the licenses on servers that are down, so LSF may have fewer licenses available after reconfiguration.

Working with Hosts

Contents

- ◆ [Host status](#) on page 53
- ◆ [How LIM Determines Host Models and Types](#) on page 55
- ◆ [Viewing Host Information](#) on page 56
- ◆ [Controlling Hosts](#) on page 61
- ◆ [Adding a Host](#) on page 63
- ◆ [Remove a Host](#) on page 66
- ◆ [Adding Hosts Dynamically](#) on page 66
- ◆ [Add Host Types and Host Models to lsf.shared](#) on page 74
- ◆ [Registering Service Ports](#) on page 75
- ◆ [Host Naming](#) on page 77
- ◆ [Hosts with Multiple Addresses](#) on page 79
- ◆ [Host Groups](#) on page 83
- ◆ [Using IPv6 Addresses](#) on page 82
- ◆ [Tuning CPU Factors](#) on page 86
- ◆ [Handling Host-level Job Exceptions](#) on page 88

Host status

Host status describes the ability of a host to accept and run batch jobs in terms of daemon states, load levels, and administrative controls. The `bhosts` and `lsload` commands display host status.

`bhosts`

Displays the current status of the host:

STATUS	Description
ok	Host is available to accept and run new batch jobs.
unavail	Host is down, or LIM and <code>sbatchd</code> are unreachable.

Host status

STATUS	Description
unreach	LIM is running but sbatchd is unreachable.
closed	Host will not accept new jobs. Use bhosts -l to display the reasons.
unlicensed	Host does not have a valid license.

bhosts -l

Displays the closed reasons. A closed host does not accept new batch jobs:

Closed reason	Description
closed_Adm	An LSF administrator or root explicitly closed the host using badmin hclose. Running jobs are not affected.
closed_Busy	The value of a load index exceeded a threshold (configured in lsb.hosts, displayed by bhosts -l). Running jobs are not affected. Indices that exceed thresholds are identified with an asterisk (*).
closed_Excl	An exclusive batch job (i.e., bsub -x) is running on the host.
closed_Full	The configured maximum number of running jobs has been reached. Running jobs will not be affected.
closed_LIM	sbatchd is running but LIM is unavailable.
closed_Lock	An LSF administrator or root explicitly locked the host using ladmin limlock. Running jobs are suspended (SSUSP). Use ladmin limunlock to unlock LIM on the local host.
closed_Wind	Host is closed by a dispatch window defined in lsb.hosts. Running jobs are not affected.
closed_EGO	For EGO-enabled SLA scheduling, closed_EGO indicates that the host is closed because it has not been allocated by EGO to run LSF jobs. Hosts allocated from EGO display status ok.

bhosts

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	-	55	2	2	0	0	0
hostB	closed	-	20	16	16	0	0	0
...								

bhosts -l hostB

HOST	hostB	STATUS	CPUF	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV	DISPATCH_WINDOW		
		closed_Adm	23.10	-	55	2	2	0	0	0	-		
CURRENT LOAD USED FOR SCHEDULING:													
			r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
Total			1.0	-0.0	-0.0	4%	9.4	148	2	3	4231M	698M	233M
Reserved			0.0	0.0	0.0	0%	0.0	0	0	0	0M	0M	0M
LOAD THRESHOLD USED FOR SCHEDULING:													
			r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched			-	-	-	-	-	-	-	-	-	-	-
loadStop			-	-	-	-	-	-	-	-	-	-	-
cpuspeed bandwidth													
loadSched			-	-	-	-	-	-	-	-	-	-	-
loadStop			-	-	-	-	-	-	-	-	-	-	-

lsload

Displays the current state of the host:

Status	Description
ok	Host is available to accept and run batch jobs and remote tasks.
-ok	LIM is running but RES is unreachable.
busy	Does not affect batch jobs, only used for remote task placement (i.e., <code>lshosts</code>). The value of a load index exceeded a threshold (configured in <code>lsf.cluster.cluster_name</code> , displayed by <code>lshosts -l</code>). Indices that exceed thresholds are identified with an asterisk (*).
lockW	Does not affect batch jobs, only used for remote task placement (i.e., <code>lshosts</code>). Host is locked by a run window (configured in <code>lsf.cluster.cluster_name</code> , displayed by <code>lshosts -l</code>).
lockU	Will not accept new batch jobs or remote tasks. An LSF administrator or root explicitly locked the host using <code>lsadmin limlock</code> , or an exclusive batch job (<code>bsub -x</code>) is running on the host. Running jobs are not affected. Use <code>lsadmin limunlock</code> to unlock LIM on the local host.
unavail	Host is down, or LIM is unavailable.
unlicensed	The host does not have a valid license.

```

lsload
HOST_NAME      status  r15s   r1m   r15m   ut    pg   ls    it    tmp    swp    mem
hostA          ok     0.0   0.0   0.0   4%   0.4  0   4316  10G   302M  252M
hostB          ok     1.0   0.0   0.0   4%   8.2  2    14  4231M  698M  232M
...

```

How LIM Determines Host Models and Types

The LIM (load information manager) daemon/service automatically collects information about hosts in an LSF cluster, and accurately determines running host models and types. At most, 1024 model types can be manually defined in `lsf.shared`.

If `lsf.shared` is not fully defined with all known host models and types found in the cluster, LIM attempts to match an unrecognized running host to one of the models and types that is defined.

LIM supports both exact matching of host models and types, and "fuzzy" matching, where an entered host model name or type is slightly different from what is defined in `lsf.conf` (or in `ego.conf` if EGO is enabled in the LSF cluster).

How does "fuzzy" matching work?

LIM reads host models and types that have been manually configured in `lsf.shared`. The format for entering host models and types is *model_bogomips_architecture* (for example, `x15_4604_0pterontmProcessor142`, `IA64_2793`, or `SUNWUltrtra510_360_sparc`). Names can be up to 64 characters long.

When LIM attempts to match running host model with what's entered in `lsf.shared`, it first attempts an exact match, then proceeds to make a fuzzy match.

How LIM attempts to make matches

Architecture name of running host	What the lim reports	Additional information about the lim process
Same as definition in <code>lsf.shared</code> (exact match)	Reports the reference index of exact match	LIM detects an exact match between model and input architecture string
Similar to what is defined in <code>lsf.shared</code> (fuzzy match)	Reports fuzzy match based on detection of 1 or 2 fields in the input architecture string	<ul style="list-style-type: none"> ◆ For input architecture strings with only one field, if LIM cannot detect an exact match for the input string, then it reports the <i>best match</i>. A best match is a model field with the most characters shared by the input string. ◆ For input architecture strings with two fields: <ul style="list-style-type: none"> a If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string b LIM then attempts to find the best match on the <i>bogomips</i> field ◆ For architecture strings with three fields: <ul style="list-style-type: none"> a If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string b After finding the best match for the model field, LIM attempts to find the best match on the <i>architecture</i> field c LIM then attempts to find the closest match on the <i>bogomips</i> field, with wildcards supported (where the <i>bogomips</i> field is a wildcard)
Has an illegal name	Reports default host model	An illegal name is one that does not follow the permitted format for entering an architecture string where the first character of the string is not an English-language character.

Viewing Host Information

LSF uses some or all of the hosts in a cluster as execution hosts. The host list is configured by the LSF administrator. Use the `bhosts` command to view host information. Use the `lsload` command to view host load information.

To view...	Run...
All hosts in the cluster and their status	<code>bhosts</code>
Condensed host groups in an uncondensed format	<code>bhosts -X</code>
Detailed server host information	<code>bhosts -l</code> and <code>lshosts -l</code>
Host load by host	<code>lsload</code>
Host architecture information	<code>lshosts</code>
Host history	<code>badmin hhist</code>
Host model and type information	<code>lsinfo</code>
Job exit rate and load for hosts	<code>bhosts -l</code> and <code>bhosts -x</code>
Dynamic host information	<code>lshosts</code>

View all hosts in the cluster and their status

- 1 Run `bhosts` to display information about all hosts and their status.

`bhosts` displays condensed information for hosts that belong to condensed host groups. When displaying members of a condensed host group, `bhosts` lists the host group name instead of the name of the individual host. For example, in a cluster with a condensed host group (`groupA`), an uncondensed host group (`groupB` containing `hostC` and `hostE`), and a host that is not in any host group (`hostF`), `bhosts` displays the following:

```
bhosts
HOST_NAME      STATUS      JL/U    MAX  NJOBS    RUN  SSUSP  USUSP  RSV
groupA         ok           5      8    4        2    0      1      1
hostC          ok           -      3    0        0    0      0      0
hostE          ok           2      4    2        1    0      0      1
hostF          ok           -      2    2        1    0      1      0
```

Define condensed host groups in the `HostGroups` section of `lsb.hosts`. To find out more about condensed host groups and to see the configuration for the above example, see [Defining condensed host groups](#) on page 85.

View uncondensed host information

- 1 Run `bhosts -X` to display all hosts in an uncondensed format, including those belonging to condensed host groups:

```
bhosts -X
HOST_NAME      STATUS      JL/U    MAX  NJOBS    RUN  SSUSP  USUSP  RSV
hostA          ok           2      2    0        0    0      0      0
hostD          ok           2      4    2        1    0      0      1
hostB          ok           1      2    2        1    0      1      0
hostC          ok           -      3    0        0    0      0      0
hostE          ok           2      4    2        1    0      0      1
hostF          ok           -      2    2        1    0      1      0
```

View detailed server host information

- 1 Run `bhosts -l host_name` and `lshosts -l host_name` to display all information about each server host such as the CPU factor and the load thresholds to start, suspend, and resume jobs:

```
bhosts -l hostB
HOST hostB
STATUS CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOWS
ok     20.20 -     -    0      0    0      0      0    -
CURRENT LOAD USED FOR SCHEDULING:
      r15s r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
Total  0.1  0.1  0.1  9%  0.7  24  17  0  394M  396M  12M
Reserved 0.0  0.0  0.0  0%  0.0  0  0  0  0M  0M  0M
```

Viewing Host Information

```
LOAD THRESHOLD USED FOR SCHEDULING:
      r15s  r1m   r15m  ut   pg   io   ls   it   tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

      cpuspeed  bandwidth
loadSched      -          -
loadStop      -          -

lshosts -l hostB
HOST_NAME: hostB
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1      2      1 2016M 1983M 72917M      0   Yes  1   2      2

RESOURCES: Not defined
RUN_WINDOWS: (always open)

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster LSF_Sched_Fairshare
LSF_Sched_Resource_Reservation LSF_Sched_Preemption LSF_Sched_Parallel
LSF_Sched_Advance_Reservation LSF_DualCore_x86)

      LICENSE CLASS NEEDED: Class(B), Multi-cores

LOAD_THRESHOLDS:
      r15s  r1m   r15m  ut   pg   io   ls   it   tmp  swp  mem
      -    1.0  -    -    -    -    -    -    -    -    4M
```

View host load by host

The `lsload` command reports the current status and load levels of hosts in a cluster.

The `lshosts -l` command shows the load thresholds.

The `lsmom` command provides a dynamic display of the load information. The LSF administrator can find unavailable or overloaded hosts with these tools.

1 Run `lsload` to see load levels for each host:

```
lsload
HOST_NAME status r15s r1m  r15m ut   pg   ls   it   tmp  swp  mem
hostD     ok      1.3 1.2  0.9 92% 0.0 2   20 5M 148M 88M
hostB     -ok     0.1 0.3  0.7 0%   0.0 1   67 45M 25M 34M
hostA     busy    8.0 *7.0 4.9 84% 4.6 6   17 1M 81M 27M
```

The first line lists the load index names, and each following line gives the load levels for one host.

Viewing host architecture (type and model) information

An LSF cluster may consist of hosts of differing architectures and speeds. The `lshosts` command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors. For example:

```
lshosts
HOST_NAME  type      model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostD      SUNSOL   SunSparc  6.0   1      64M    112M    Yes    (solaris cserver)
hostM      RS6K     IBM350   7.0   1      64M    124M    Yes    (cserver aix)
hostC      SGI6     R10K    14.0  16     1024M  1896M   Yes    (irix cserver)
hostA      HPPA     HP715   6.0   1      98M    200M    Yes    (hpux fserver)
```

In the above example, the host type `SUNSOL` represents Sun SPARC systems running Solaris, and `SGI6` represents an SGI server running IRIX 6. The `lshosts` command also displays the resources available on each host.

type

The host CPU architecture. Hosts that can run the same binary programs should have the same type.

An `UNKNOWN` type or model indicates the host is down, or LIM on the host is down. See [UNKNOWN host type or model](#) on page 677 for instructions on measures to take.

When automatic detection of host type or model fails (the host type configured in `lsf.shared` cannot be found), the type or model is set to `DEFAULT`. LSF will work on the host, but a `DEFAULT` model may be inefficient because of incorrect CPU factors. A `DEFAULT` type may also cause binary incompatibility because a job from a `DEFAULT` host type can be migrated to another `DEFAULT` host type. automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found.

View host history

- 1 Run `badadmin hhist` to view the history of a host such as when it is opened or closed:

```
badadmin hhist hostB
Wed Nov 20 14:41:58: Host <hostB> closed by administrator <lsf>.
Wed Nov 20 15:23:39: Host <hostB> opened by administrator <lsf>.
```

View host model and type information

- 1 Run `lsinfo -m` to display information about host models that exist in the cluster:

```
lsinfo -m
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
PC1133          23.10           x6_1189_PentiumIIICoppermine
HP9K735         4.50            HP9000735_125
HP9K778         5.50            HP9000778
Ultra5S         10.30           SUNWUltra510_270_sparcv9
Ultra2          20.20           SUNWUltra2_300_sparc
Enterprise3000  20.00           SUNWUltraEnterprise_167_sparc
```

Viewing Host Information

- 2 Run `lsinfo -M` to display all host models defined in `lsf.shared`:

```
lsinfo -M
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
UNKNOWN_AUTO_DETECT  1.00      UNKNOWN_AUTO_DETECT
DEFAULT         1.00
LINUX133        2.50      x586_53_Pentium75
PC200           4.50      i86pc_200
Intel_IA64      12.00     ia64
Ultra5S         10.30     SUNWUltra5_270_sparcv9
PowerPC_G4      12.00     x7400G4
HP300           1.00
SunSparc        12.00
```

- 3 Run `lim -t` to display the type, model, and matched type of the current host. You must be the LSF administrator to use this command:

```
lim -t hostA
Host Type          : NTX64
Host Architecture  : EM64T_1596
Physical Processors : 2
Cores per Processor : 4
Threads per Core   : 2
License Needed     : Class(B), Multi-core
Matched Type       : NTX64
Matched Architecture : EM64T_3000
Matched Model      : Intel_EM64T
CPU Factor         : 60.0
```

View job exit rate and load for hosts

- 1 Run `bhosts` to display the exception threshold for job exit rate and the current load value for hosts.:

In the following example, `EXIT_RATE` for `hostA` is configured as 4 jobs per minute. `hostA` does not currently exceed this rate

```
bhosts -l hostA
HOST hostA
STATUS      CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
ok          18.60  -    1    0      0    0      0    0    -

CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
Total    0.0   0.0  0.0   0%   0.0   0    1    2   646M  648M  115M
Reserved 0.0   0.0  0.0   0%   0.0   0    0    0    0M   0M   0M

          share_rsrc  host_rsrc
Total          3.0    2.0
Reserved       0.0    0.0

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -
```

```

cpuspeed    bandwidth
loadSched   -          -
loadStop    -          -

THRESHOLD AND LOAD USED FOR EXCEPTIONS:
JOB_EXIT_RATE
Threshold   4.00
Load        0.00

```

- 2 Use `bhosts -x` to see hosts whose job exit rate has exceeded the threshold for longer than `JOB_EXIT_RATE_DURATION`, and are still high. By default, these hosts are closed the next time LSF checks host exceptions and invokes `eadmin`.

If no hosts exceed the job exit rate, `bhosts -x` displays:

```
There is no exceptional host found
```

View dynamic host information

- 1 Use `lshosts` to display information on dynamically added hosts.

An LSF cluster may consist of static and dynamic hosts. The `lshosts` command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors. Server represents the type of host in the cluster. “Yes” is displayed for LSF servers, “No” is displayed for LSF clients, and “Dyn” is displayed for dynamic hosts.

For example:

```
lshosts
HOST_NAME  type    model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA      SOL64  Ultra60F  23.5  1      64M     112M    Yes    ()
hostB      LINUX86 Opteron8  60.0  1      94M     168M    Dyn    ()
```

In the above example, `hostA` is a static host while `hostB` is a dynamic host.

Controlling Hosts

Hosts are opened and closed by an LSF Administrator or root issuing a command or through configured dispatch windows.

Close a host

- 1 Run `badmin hclose`:


```
badmin hclose hostB
Close <hostB> ..... done
```

If the command fails, it may be because the host is unreachable through network problems, or because the daemons on the host are not running.

Open a host

- 1 Run `badadmin hopen`:

```
badadmin hopen hostB
Open <hostB> ..... done
```
-

Configure Dispatch Windows

A dispatch window specifies one or more time periods during which a host will receive new jobs. The host will not receive jobs outside of the configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, dispatch windows are not configured.

To configure dispatch windows:

- 1 Edit `lsb.hosts`.
 - 2 Specify one or more time windows in the `DISPATCH_WINDOW` column:

```
Begin Host
HOST_NAME      r1m      pg      ls      tmp      DISPATCH_WINDOW
...
hostB          3.5/4.5  15/     12/15   0        (4:30-12:00)
...
End Host
```
 - 3 Reconfigure the cluster:
 - a Run `lsadmin reconfig` to reconfigure LIM.
 - b Run `badadmin reconfig` to reconfigure `mbatchd`.
 - 4 Run `bhosts -l` to display the dispatch windows.
-

Log a comment when closing or opening a host

- 1 Use the `-C` option of `badadmin hclose` and `badadmin hopen` to log an administrator comment in `lsb.events`:

```
badadmin hclose -C "Weekly backup" hostB
```

The comment text `Weekly backup` is recorded in `lsb.events`. If you close or open a host group, each host group member displays with the same comment string.

A new event record is recorded for each host open or host close event. For example:

```
badadmin hclose -C "backup" hostA
```

followed by

```
badadmin hclose -C "Weekly backup" hostA
```

generates the following records in `lsb.events`:

```
"HOST_CTRL" "7.0 1050082346 1 "hostA" 32185 "lsfadmin" "backup"
"HOST_CTRL" "7.0 1050082373 1 "hostA" 32185 "lsfadmin" "Weekly backup"
```

- 2 Use `badmin hist` or `badmin hhist` to display administrator comments for closing and opening hosts:

`badmin hhist`

```
Fri Apr 4 10:35:31: Host <hostB> closed by administrator
<lsfadmin> Weekly backup.
```

`bhosts -l` also displays the comment text:

```
bhosts -l
```

```
HOST hostA
STATUS      CPUF  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV  DISPATCH_WINDOW
closed_Adm 1.00  -      -    0        0      0      0      0      -
```

CURRENT LOAD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
Total	0.0	0.0	0.0	2%	0.0	64	2	11	7117M	512M	432M
Reserved	0.0	0.0	0.0	0%	0.0	0	0	0	0M	0M	0M

LOAD THRESHOLD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

cpuspeed bandwidth

loadSched	-	-
loadStop	-	-

THRESHOLD AND LOAD USED FOR EXCEPTIONS:

```
JOB_EXIT_RATE
Threshold 2.00
Load      0.00
```

```
ADMIN ACTION COMMENT: "Weekly backup"
```

How events are displayed and recorded in MultiCluster lease model

In the MultiCluster resource lease model, host control administrator comments are recorded only in the `lsb.events` file on the local cluster. `badmin hist` and `badmin hhist` display only events that are recorded locally. Host control messages are not passed between clusters in the MultiCluster lease model. For example, if you close an exported host in both the consumer and the provider cluster, the host close events are recorded separately in their local `lsb.events`.

Adding a Host

You use the `lsfinstall` command to add a host to an LSF cluster.

Contents

- ◆ [Add a host of an existing type using `lsfinstall` on page 64](#)

- ◆ [Add a host of a new type using lsfinstall](#) on page 65

Add a host of an existing type using lsfinstall

RESTRICTION: `lsfinstall` is not compatible with clusters installed with `lsfsetup`. To add a host to a cluster originally installed with `lsfsetup`, you must upgrade your cluster.

- 1 Verify that the host type already exists in your cluster:
 - a Log on to any host in the cluster. You do not need to be root.
 - b List the contents of the `LSF_TOP/7.0` directory. The default is `/usr/share/lsf/7.0`. If the host type currently exists, there is a subdirectory with the name of the host type. If it does not exist, go to [Add a host of a new type using lsfinstall](#) on page 65.
- 2 Add the host information to `lsf.cluster.cluster_name`:
 - a Log on to the LSF master host as root.
 - b Edit `LSF_CONFDIR/lsf.cluster.cluster_name`, and specify the following in the `Host` section:
 - ◆ The name of the host.
 - ◆ The model and type, or specify `!` to automatically detect the type or model.
 - ◆ Specify `1` for LSF server or `0` for LSF client.

```
Begin Host
HOSTNAME  model  type      server  r1m  mem  RESOURCES  REXPRI
hosta     !      SUNSOL6   1       1.0  4    ()          0
hostb     !      SUNSOL6   0       1.0  4    ()          0
hostc     !      HPPA1132  1       1.0  4    ()          0
hostd     !      HPPA1164  1       1.0  4    ()          0
End Host
```

- c Save your changes.
- 3 Run `lsadmin reconfig` to reconfigure LIM.
- 4 Run `badmin mbdrestart` to restart `mbatchd`.
- 5 Run `hostsetup` to set up the new host and configure the daemons to start automatically at boot from `/usr/share/lsf/7.0/install`:


```
./hostsetup --top="/usr/share/lsf" --boot="y"
```
- 6 Start LSF on the new host:


```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```
- 7 Run `bhosts` and `lshosts` to verify your changes.
 - ◆ If any host type or host model is UNKNOWN, follow the steps in [UNKNOWN host type or model](#) on page 677 to fix the problem.
 - ◆ If any host type or host model is DEFAULT, follow the steps in [DEFAULT host type or model](#) on page 677 to fix the problem.

Add a host of a new type using `lsfinstall`

RESTRICTION: `lsfinstall` is not compatible with clusters installed with `lsfsetup`. To add a host to a cluster originally installed with `lsfsetup`, you must upgrade your cluster.

- 1 Verify that the host type does not already exist in your cluster:
 - a Log on to any host in the cluster. You do not need to be root.
 - b List the contents of the `LSF_TOP/7.0` directory. The default is `/usr/share/lsf/7.0`. If the host type currently exists, there will be a subdirectory with the name of the host type. If the host type already exists, go to [Add a host of an existing type using `lsfinstall`](#) on page 64.
- 2 Get the LSF distribution tar file for the host type you want to add.
- 3 Log on as root to any host that can access the LSF install directory.
- 4 Change to the LSF install directory. The default is `/usr/share/lsf/7.0/install`
- 5 Edit `install.config`:
 - a For `LSF_TARDIR`, specify the path to the tar file. For example:


```
LSF_TARDIR="/usr/share/lsf_distrib/7.0"
```
 - b For `LSF_ADD_SERVERS`, list the new host names enclosed in quotes and separated by spaces. For example:


```
LSF_ADD_SERVERS="hosta hostb"
```
 - c Run `./lsfinstall -f install.config`. This automatically creates the host information in `lsf.cluster.cluster_name`.
- 6 Run `lsadmin reconfig` to reconfigure LIM.
- 7 Run `badmin reconfig` to reconfigure `mbatchd`.
- 8 Run `hostsetup` to set up the new host and configure the daemons to start automatically at boot from `/usr/share/lsf/7.0/install`:


```
./hostsetup --top="/usr/share/lsf" --boot="y"
```
- 9 Start LSF on the new host:


```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```
- 10 Run `bhosts` and `lshosts` to verify your changes.
 - ◆ If any host type or host model is UNKNOWN, follow the steps in [UNKNOWN host type or model](#) on page 677 to fix the problem.
 - ◆ If any host type or host model is DEFAULT, follow the steps in [DEFAULT host type or model](#) on page 677 to fix the problem.

Remove a Host

Removing a host from LSF involves preventing any additional jobs from running on the host, removing the host from LSF, and removing the host from the cluster.

CAUTION: Never remove the master host from LSF. If you want to remove your current default master from LSF, change `lsf.cluster.cluster_name` to assign a different default master host. Then remove the host that was once the master host.

- 1 Log on to the LSF host as root.
 - 2 Run `badmin hclose` to close the host. This prevents jobs from being dispatched to the host and allows running jobs to finish.
 - 3 When all dispatched jobs are finished, run `lsfshutdown` to stop the LSF daemons.
 - 4 Remove any references to the host in the Host section of `LSF_CONFDIR/lsf.cluster.cluster_name`.
 - 5 Remove any other references to the host, if applicable, from the following LSF configuration files:
 - ◆ `LSF_CONFDIR/lsf.shared`
 - ◆ `LSB_CONFDIR/lsb.hosts`
 - ◆ `LSB_CONFDIR/lsb.queues`
 - 6 Log off the host to be removed, and log on as `root` or the primary LSF administrator to any other host in the cluster.
 - 7 Run `lsadmin reconfig` to reconfigure LIM.
 - 8 Run `badmin mbdrestart` to restart `mbatchd`.
 - 9 If you configured LSF daemons to start automatically at system startup, remove the LSF section from the host's system startup files.
 - 10 If any users of the host use `lstcsh` as their login shell, change their login shell to `tcsh` or `csh`. Remove `lstcsh` from the `/etc/shells` file.
-

Adding Hosts Dynamically

By default, all configuration changes made to LSF are static. To add or remove hosts within the cluster, you must manually change the configuration and restart all master candidates.

Dynamic host configuration allows you to add and remove hosts without manual reconfiguration. To enable dynamic host configuration, all of the parameters described in the following table must be defined.

Parameter	Defined in ...	Description
LSF_MASTER_LIST	<code>lsf.conf</code>	Defines a list of master host candidates. These hosts receive information when a dynamic host is added to or removed from the cluster. Do not add dynamic hosts to this list, because dynamic hosts cannot be master hosts.
LSF_DYNAMIC_HOST_WAIT_TIME	<code>lsf.conf</code>	Defines the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.
LSF_HOST_ADDR_RANGE	<code>lsf.cluster.cluster_name</code>	Identifies the range of IP addresses for hosts that can dynamically join or leave the cluster.

IMPORTANT: If you choose to enable dynamic hosts when you install LSF, the installer adds the parameter `LSF_HOST_ADDR_RANGE` to `lsf.cluster.cluster_name` using a default value that allows any host to join the cluster. To enable security, configure `LSF_HOST_ADDR_RANGE` in `lsf.cluster.cluster_name` after installation to restrict the hosts that can join your cluster.

How dynamic host configuration works

Master LIM The master LIM runs on the master host for the cluster. The master LIM receives requests to add hosts, and tells the master host candidates defined by the parameter `LSF_MASTER_LIST` to update their configuration information when a host is dynamically added or removed.

Upon startup, both static and dynamic hosts wait to receive an acknowledgement from the master LIM. This acknowledgement indicates that the master LIM has added the host to the cluster. Static hosts normally receive an acknowledgement because the master LIM has access to static host information in the LSF configuration files. Dynamic hosts do not receive an acknowledgement, however, until they announce themselves to the master LIM. The parameter `LSF_DYNAMIC_HOST_WAIT_TIME` in `lsf.conf` determines how long a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

Master candidate LIMs The parameter `LSF_MASTER_LIST` defines the list of master host candidates. These hosts receive updated host information from the master LIM so that any master host candidate can take over as master host for the cluster.

IMPORTANT: Master candidate hosts should share LSF configuration and binaries.

Dynamic hosts cannot be master host candidates. By defining the parameter `LSF_MASTER_LIST`, you ensure that LSF limits the list of master host candidates to specific, static hosts.

Adding Hosts Dynamically

mbatchd `mbatchd` gets host information from the master LIM; when it detects the addition or removal of a dynamic host within the cluster, `mbatchd` automatically reconfigures itself.

TIP: After adding a host dynamically, you might have to wait for `mbatchd` to detect the host and reconfigure. Depending on system load, `mbatchd` might wait up to a maximum of 10 minutes before reconfiguring.

lsadmin command Use the command `lsadmin limstartup` to start the LIM on a newly added dynamic host.

Allowing only certain hosts to join the cluster

By default, any host can be dynamically added to the cluster. To enable security, define `LSF_HOST_ADDR_RANGE` in `lsf.cluster.cluster_name` to identify a range of IP addresses for hosts that are allowed to dynamically join the cluster as LSF hosts. IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`; you do not have to map IPv4 addresses to an IPv6 format.

Configure LSF to run batch jobs on dynamic hosts

Before you run batch jobs on a dynamic host, complete any or all of the following steps, depending on your cluster configuration.

-
- 1 Configure queues to accept all hosts by defining the `HOSTS` parameter in `lsb.queues` using the keyword `all`.
 - 2 Define host groups that will accept wild cards in the `HostGroup` section of `lsb.hosts`.
For example, define `linuxrack*` as a `GROUP_MEMBER` within a host group definition.
 - 3 Add a dynamic host to a host group using the command `badmin hghostadd`.
-

Changing a dynamic host to a static host

If you want to change a dynamic host to a static host, first use the command `badmin hghostdel` to remove the dynamic host from any host group that it belongs to, and then configure the host as a static host in `lsf.cluster.cluster_name`.

Adding dynamic hosts

Add a dynamic host in a shared file system environment

In a shared file system environment, you do not need to install LSF on each dynamic host. The master host will recognize a dynamic host as an LSF host when you start the daemons on the dynamic host.

- 1 In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_WAIT_TIME`, in seconds, and assign a value greater than zero.

`LSF_DYNAMIC_HOST_WAIT_TIME` specifies the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

- 2 In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_TIMEOUT`.

`LSF_DYNAMIC_HOST_TIMEOUT` specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

NOTE: For very large clusters, defining this parameter could decrease system performance.

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

- 3 In `lsf.cluster.cluster_name` on the master host, define the parameter `LSF_HOST_ADDR_RANGE`.

`LSF_HOST_ADDR_RANGE` enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

TIP: If you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`, IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format; you do not have to map IPv4 addresses to an IPv6 format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access. In this example, no IPv6 hosts are allowed.

- 4 Log on as root to each host you want to join the cluster.
- 5 Source the LSF environment:

Adding Hosts Dynamically

- ❖ For `csh` or `tcsh`:
`source LSF_TOP/conf/cshrc.lsf`
 - ❖ For `sh`, `ksh`, or `bash`:
`. LSF_TOP/conf/profile.lsf`
- 6 Do you want LSF to start automatically when the host reboots?
- ❖ If no, go to step 7.
 - ❖ If yes, run the `hostsetup` command. For example:
`# cd /usr/share/lsf/7.0/install`
`# ./hostsetup --top="/usr/share/lsf" --boot="y"`
For complete `hostsetup` usage, enter `hostsetup -h`.
- 7 Use the following commands to start LSF:
- ```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```
- 

## Add a dynamic host in a non-shared file system environment

In a non-shared file system environment, you must install LSF binaries, a localized `lsf.conf` file, and shell environment scripts (`cshrc.lsf` and `profile.lsf`) on each dynamic host.

### Specify installation options in the `slave.config` file

All dynamic hosts are slave hosts, because they cannot serve as master host candidates. The `slave.config` file contains parameters for configuring all slave hosts.

- 1 Define the required parameters.  
`LSF_SERVER_HOSTS="host_name [host_name ...]"`  
`LSF_ADMINS="user_name [ user_name ... ]"`  
`LSF_TOP="/path"`
- 2 Define the optional parameters.  
`LSF_LIM_PORT=port_number`

---

**IMPORTANT:** If the master host does not use the default `LSF_LIM_PORT`, you must specify the same `LSF_LIM_PORT` defined in `lsf.conf` on the master host.

---

## Add local resources on a dynamic host to the cluster

**Prerequisites:** Ensure that the resource name and type are defined in `lsf.shared`, and that the ResourceMap section of `lsf.cluster.cluster_name` contains at least one resource mapped to at least one static host. LSF can add local resources as long as the ResourceMap section is defined; you do not need to map the local resources.

- 1 In the `slave.config` file, define the parameter `LSF_LOCAL_RESOURCES`.

For numeric resources, define name-value pairs:

```
"[resourcemap value*resource_name]"
```

For Boolean resources, the value is the resource name in the following format:

```
"[resource resource_name]"
```

For example:

```
LSF_LOCAL_RESOURCES="[resourcemap 1*verilog] [resource linux]"
```

**TIP:** If `LSF_LOCAL_RESOURCES` are already defined in a local `lsf.conf` on the dynamic host, `lsfinstall` does not add resources you define in `LSF_LOCAL_RESOURCES` in `slave.config`.

When the dynamic host sends a request to the master host to add it to the cluster, the dynamic host also reports its local resources. If the local resource is already defined in `lsf.cluster.cluster_name` as `default` or `all`, it cannot be added as a local resource.

## Install LSF on a dynamic host

- 1 Run `lsfinstall -s -f slave.config`.

`lsfinstall` creates a local `lsf.conf` for the dynamic host, which sets the following parameters:

```
LSF_CONFDIR="/path"
```

```
LSF_GET_CONF=lim
```

```
LSF_LIM_PORT=port_number (same as the master LIM port number)
```

```
LSF_LOCAL_RESOURCES="resource ..."
```

**TIP:** Do not duplicate `LSF_LOCAL_RESOURCES` entries in `lsf.conf`. If local resources are defined more than once, only the last definition is valid.

```
LSF_SERVER_HOSTS="host_name [host_name ...]"
```

```
LSF_VERSION=7
```

**IMPORTANT:** If `LSF_STRICT_CHECKING` is defined in `lsf.conf` to protect your cluster in untrusted environments, and your cluster has dynamic hosts, `LSF_STRICT_CHECKING` must be configured in the local `lsf.conf` on all dynamic hosts.

## Configure dynamic host parameters

- 1 In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_WAIT_TIME`, in seconds, and assign a value greater than zero.

`LSF_DYNAMIC_HOST_WAIT_TIME` specifies the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

- 2 In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_TIMEOUT`.

`LSF_DYNAMIC_HOST_TIMEOUT` specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

**NOTE:** For very large clusters, defining this parameter could decrease system performance.

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

- 3 In `lsf.cluster.cluster_name` on the master host, define the parameter `LSF_HOST_ADDR_RANGE`.

`LSF_HOST_ADDR_RANGE` enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

**TIP:** If you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`, IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format; you do not have to map IPv4 addresses to an IPv6 format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access. No IPv6 hosts are allowed.

## Start LSF daemons

- 1 Log on as root to each host you want to join the cluster.
- 2 Source the LSF environment:
  - ❖ For `csh` or `tcsh`:

```
source LSF_TOP/conf/cshrc.lsf
```
  - ❖ For `sh`, `ksh`, or `bash`:

```
. LSF_TOP/conf/profile.lsf
```



- 3 Do you want LSF to start automatically when the host reboots?
  - ❖ If no, go to step 4.
  - ❖ If yes, run the `hostsetup` command. For example:
 

```
cd /usr/share/lsf/7.0/install
./hostsetup --top="/usr/share/lsf" --boot="y"
```

 For complete `hostsetup` usage, enter `hostsetup -h`.
- 4 Is this the first time the host is joining the cluster?
  - ❖ If no, use the following commands to start LSF:
 

```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```
  - ❖ If yes, you must start the daemons from the local host. For example, if you want to start the daemons on `hostB` from `hostA`, use the following commands:
 

```
rsh hostB lsadmin limstartup
rsh hostB lsadmin resstartup
rsh hostB badmin hstartup
```

## Removing dynamic hosts

To remove a dynamic host from the cluster, you can either set a timeout value, or you can edit the `hostcache` file.

### Remove a host by setting a timeout value

`LSF_DYNAMIC_HOST_TIMEOUT` specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

**NOTE:** For very large clusters, defining this parameter could decrease system performance. If you want to use this parameter to remove dynamic hosts from a very large cluster, disable the parameter after LSF has removed the unwanted hosts.

- 1 In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_TIMEOUT`.  
To specify minutes rather than hours, append `m` or `M` to the value.  
For example:
 

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

## Remove a host by editing the hostcache file

Dynamic hosts remain in the cluster unless you intentionally remove them. Only the cluster administrator can modify the `hostcache` file.

- 1 Shut down the cluster.

```
#lsfshutdown
```

This shuts down LSF on all hosts in the cluster and prevents LIMs from trying to write to the `hostcache` file while you edit it.

- 2 In the file `$EGO_WORKDIR/lim/hostcache`, delete the line for the dynamic host that you want to remove.

- 3 Close the `hostcache` file, and then start up the cluster.

```
#lsfrestart
```

## Add Host Types and Host Models to lsf.shared

The `lsf.shared` file contains a list of host type and host model names for most operating systems. You can add to this list or customize the host type and host model names. A host type and host model name can be any alphanumeric string up to 39 characters long.

### Add a custom host type or model

- 1 Log on as the LSF administrator on any host in the cluster.

- 2 Edit `lsf.shared`:

- a For a new host type, modify the `HostType` section:

```
Begin HostType
TYPENAME # Keyword
DEFAULT
IBMAIX564
LINUX86
LINUX64
NTX64
NTIA64
SUNSOL
SOL732
SOL64
SGI658
SOLX86
HPPA11
HPUXIA64
MACOSX
End HostType
```

- b For a new host model, modify the `HostModel` section:

Add the new model and its CPU speed factor relative to other models. For more details on tuning CPU factors, see [Tuning CPU Factors](#) on page 86.

```
Begin HostModel
MODELNAME CPUFACTOR ARCHITECTURE # keyword
x86 (Solaris, Windows, Linux): approximate values, based on SpecBench results
for Intel processors (Sparc/Win) and BogoMIPS results (Linux).
PC75 1.5 (i86pc_75 i586_75 x586_30)
PC90 1.7 (i86pc_90 i586_90 x586_34 x586_35 x586_36)
HP9K715 4.2 (HP9000715_100)
SunSparc 12.0 ()
CRAYJ90 18.0 ()
IBM350 18.0 ()
End HostModel
```

- 3 Save the changes to `lsf.shared`.
- 4 Run `lsadmin reconfig` to reconfigure LIM.
- 5 Run `badmin reconfig` to reconfigure `mbatchd`.

## Registering Service Ports

LSF uses dedicated UDP and TCP ports for communication. All hosts in the cluster must use the same port numbers to communicate with each other.

The service port numbers can be any numbers ranging from 1024 to 65535 that are not already used by other services. To make sure that the port numbers you supply are not already used by applications registered in your service database check `/etc/services` or use the command `yycat services`.

By default, port numbers for LSF services are defined in the `lsf.conf` file. You can also configure ports by modifying `/etc/services` or the NIS or NIS+ database. If you define port numbers `lsf.conf`, port numbers defined in the service database are ignored.

### lsf.conf

- 1 Log on to any host as `root`.
- 2 Edit `lsf.conf` and add the following lines:

```
LSF_RES_PORT=3878
LSB_MBD_PORT=3881
LSB_SBD_PORT=3882
```
- 3 Add the same entries to `lsf.conf` on every host.
- 4 Save `lsf.conf`.
- 5 Run `lsadmin reconfig` to reconfigure LIM.
- 6 Run `badmin mbdrestart` to restart `mbatchd`.
- 7 Run `lsfstartup` to restart all daemons in the cluster.

### /etc/services

#### Configure services manually

---

**TIP:** During installation, use the `hostsetup --boot="y"` option to set up the LSF port numbers in the service database.

---

- 1 Use the file `LSF_TOP/version/install/instlib/example.services` file as a guide for adding LSF entries to the services database.

If any other service listed in your services database has the same port number as one of the LSF services, you must change the port number for the LSF service. You must use the same port numbers on every LSF host.

- 2 Log on to any host as `root`.
- 3 Edit the `/etc/services` file by adding the contents of the `LSF_TOP/version/install/instlib/example.services` file:

```
/etc/services entries for LSF daemons
#
res 3878/tcp # remote execution server
lim 3879/udp # load information manager
mbatchd 3881/tcp # master lsbatch daemon
sbatchd 3882/tcp # slave lsbatch daemon
#
Add this if ident is not already defined
in your /etc/services file
ident 113/tcp auth tap # identd
```

- 4 Run `lsadmin reconfig` to reconfigure LIM.
  - 5 Run `badmin reconfig` to reconfigure `mbatchd`.
  - 6 Run `lsfstartup` to restart all daemons in the cluster.
- 

#### NIS or NIS+ database

If you are running NIS, you only need to modify the services database once per NIS master. On some hosts the NIS database and commands are in the `/var/yp` directory; on others, NIS is found in `/etc/yp`.

- 
- 1 Log on to any host as `root`.
  - 2 Run `lsfshutdown` to shut down all the daemons in the cluster
  - 3 To find the name of the NIS master host, use the command:  
`ypwhich -m services`
  - 4 Log on to the NIS master host as `root`.

- 5 Edit the `/var/yp/src/services` or `/etc/yp/src/services` file on the NIS master host adding the contents of the `LSF_TOP/version/install/instlib/example.services` file:

```
/etc/services entries for LSF daemons.
#
res 3878/tcp # remote execution server
lim 3879/udp # load information manager
mbatchd 3881/tcp # master lsbatch daemon
sbatchd 3882/tcp # slave lsbatch daemon
#
Add this if ident is not already defined
in your /etc/services file
ident 113/tcp auth tap # identd
```

Make sure that all the lines you add either contain valid service entries or begin with a comment character (`#`). Blank lines are not allowed.

- 6 Change the directory to `/var/yp` or `/etc/yp`.

- 7 Use the following command:

```
ypmake services
```

On some hosts the master copy of the services database is stored in a different location.

On systems running NIS+ the procedure is similar. Refer to your system documentation for more information.

- 8 Run `lsadmin reconfig` to reconfigure LIM.

- 9 Run `badmin reconfig` to reconfigure `mbatchd`.

- 10 Run `lsfstartup` to restart all daemons in the cluster.

## Host Naming

LSF needs to match host names with the corresponding Internet host addresses.

LSF looks up host names and addresses the following ways:

- ◆ In the `/etc/hosts` file
- ◆ Sun Network Information Service/Yellow Pages (NIS or YP)
- ◆ Internet Domain Name Service (DNS).

DNS is also known as the Berkeley Internet Name Domain (BIND) or `named`, which is the name of the BIND daemon.

Each host is configured to use one or more of these mechanisms.

## Network addresses

Each host has one or more network addresses; usually one for each network to which the host is directly connected. Each host can also have more than one name.

**Official host name** The first name configured for each address is called the official name.

**Host name aliases** Other names for the same host are called aliases.

LSF uses the configured host naming system on each host to look up the official host name for any alias or host address. This means that you can use aliases as input to LSF, but LSF always displays the official name.

### Using host name ranges as aliases

The default host file syntax

```
ip_address official_name [alias [alias ...]]
```

is powerful and flexible, but it is difficult to configure in systems where a single host name has many aliases, and in multihomed host environments.

In these cases, the `hosts` file can become very large and unmanageable, and configuration is prone to error.

The syntax of the LSF `hosts` file supports host name ranges as aliases for an IP address. This simplifies the host name alias specification.

To use host name ranges as aliases, the host names must consist of a fixed node group name prefix and node indices, specified in a form like:

```
host_name[index_x-index_y, index_m, index_a-index_b]
```

For example:

```
atlasD0[0-3,4,5-6, ...]
```

is equivalent to:

```
atlasD0[0-6, ...]
```

The node list does not need to be a continuous range (some nodes can be configured out). Node indices can be numbers or letters (both upper case and lower case).

**Example** Some systems map internal compute nodes to single LSF host names. A host file might contain 64 lines, each specifying an LSF host name and 32 node names that correspond to each LSF host:

```
...
177.16.1.1 atlasD0 atlas0 atlas1 atlas2 atlas3 atlas4 ... atlas31
177.16.1.2 atlasD1 atlas32 atlas33 atlas34 atlas35 atlas36 ... atlas63
...
```

In the new format, you still map the nodes to the LSF hosts, so the number of lines remains the same, but the format is simplified because you only have to specify ranges for the nodes, not each node individually as an alias:

```
...
177.16.1.1 atlasD0 atlas[0-31]
177.16.1.2 atlasD1 atlas[32-63]
...
```

You can use either an IPv4 or an IPv6 format for the IP address (if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`).

### Host name services

#### Digital UNIX

On Digital Unix systems, the `/etc/svc.conf` file controls which host name service is used.

## Solaris

On Solaris systems, the `/etc/nsswitch.conf` file controls the name service.

## Other UNIX platforms

On other UNIX platforms, the following rules apply:

- ◆ If your host has an `/etc/resolv.conf` file, your host is using DNS for name lookups
- ◆ If the command `yycat hosts` prints out a list of host addresses and names, your system is looking up names in NIS
- ◆ Otherwise, host names are looked up in the `/etc/hosts` file

## For more information

The man pages for the `gethostbyname` function, the `ybind` and `named` daemons, the `resolver` functions, and the `hosts`, `svc.conf`, `nsswitch.conf`, and `resolv.conf` files explain host name lookups in more detail.

# Hosts with Multiple Addresses

## Multi-homed hosts

Hosts that have more than one network interface usually have one Internet address for each interface. Such hosts are called *multi-homed hosts*. For example, dual-stack hosts are multi-homed because they have both an IPv4 and an IPv6 network address.

LSF identifies hosts by name, so it needs to match each of these addresses with a single host name. To do this, the host name information must be configured so that all of the Internet addresses for a host resolve to the same name.

There are two ways to do it:

- ◆ Modify the system hosts file (`/etc/hosts`) and the changes will affect the whole system
- ◆ Create an LSF hosts file (`LSF_CONFDIR/hosts`) and LSF will be the only application that resolves the addresses to the same host

## Multiple network interfaces

Some system manufacturers recommend that each network interface, and therefore, each Internet address, be assigned a different host name. Each interface can then be directly accessed by name. This setup is often used to make sure NFS requests go to the nearest network interface on the file server, rather than going through a router to some other interface. Configuring this way can confuse LSF, because there is no way to determine that the two different names (or addresses) mean the same host. LSF provides a workaround for this problem.

All host naming systems can be configured so that host address lookups always return the same name, while still allowing access to network interfaces by different names. Each host has an official name and a number of aliases, which are other names for the same host. By configuring all interfaces with the same official name but different aliases, you can refer to each interface by a different alias name while still providing a single official name for the host.

## Configuring the LSF hosts file

If your LSF clusters include hosts that have more than one interface and are configured with more than one official host name, you must either modify the host name configuration, or create a private `hosts` file for LSF to use.

The LSF `hosts` file is stored in `LSF_CONFDIR`. The format of `LSF_CONFDIR/hosts` is the same as for `/etc/hosts`.

In the LSF `hosts` file, duplicate the system `hosts` database information, except make all entries for the host use the same official name. Configure all the other names for the host as aliases so that people can still refer to the host by any name.

### Example

For example, if your `/etc/hosts` file contains:

```
AA.AA.AA.AA host-AA host # first interface
BB.BB.BB.BB host-BB # second interface
```

then the `LSF_CONFDIR/hosts` file should contain:

```
AA.AA.AA.AA host host-AA # first interface
BB.BB.BB.BB host host-BB # second interface
```

## Example `/etc/hosts` entries

### No unique official name

The following example is for a host with two interfaces, where the host does not have a unique official name.

```
Address Official name Aliases
Interface on network A
AA.AA.AA.AA host-AA.domain host.domain host-AA host
Interface on network B
BB.BB.BB.BB host-BB.domain host-BB host
```

Looking up the address `AA.AA.AA.AA` finds the official name `host-AA.domain`. Looking up address `BB.BB.BB.BB` finds the name `host-BB.domain`. No information connects the two names, so there is no way for LSF to determine that both names, and both addresses, refer to the same host.

To resolve this case, you must configure these addresses using a unique host name. If you cannot make this change to the system file, you must create an LSF `hosts` file and configure these addresses using a unique host name in that file.

### Both addresses have the same official name

Here is the same example, with both addresses configured for the same official name.

```
Address Official name Aliases
Interface on network A
AA.AA.AA.AA host.domain host-AA.domain host-AA host
Interface on network B
BB.BB.BB.BB host.domain host-BB.domain host-BB host
```



With this configuration, looking up either address returns `host.domain` as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the `host-AA` and `host-BB` aliases.

### Example for a dual-stack host

Dual-stack hosts have more than one IP address. You must associate the host name with both addresses, as shown in the following example:

```
Address Official name Aliases
Interface IPv4
AA.AA.AA.AA host.domain host-AA.domain
Interface IPv6
BBBB:BBBB:BBBB:BBBB:BBBB:BBBB::BBBB host.domain host-BB.domain
```

With this configuration, looking up either address returns `host.domain` as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the `host-AA` and `host-BB` aliases.

### Sun Solaris example

For example, Sun NIS uses the `/etc/hosts` file on the NIS master host as input, so the format for NIS entries is the same as for the `/etc/hosts` file. Since LSF can resolve this case, you do not need to create an LSF hosts file.

### DNS configuration

The configuration format is different for DNS. The same result can be produced by configuring two address (A) records for each Internet address. Following the previous example:

```
name class type address
host.domain IN A AA.AA.AA.AA
host.domain IN A BB.BB.BB.BB
host-AA.domain IN A AA.AA.AA.AA
host-BB.domain IN A BB.BB.BB.BB
```

Looking up the official host name can return either address. Looking up the interface-specific names returns the correct address for each interface.

For a dual-stack host:

```
name class type address
host.domain IN A AA.AA.AA.AA
host.domain IN A BBBB:BBBB:BBBB:BBBB:BBBB:BBBB::BBBB
host-AA.domain IN A AA.AA.AA.AA
host-BB.domain IN A BBBB:BBBB:BBBB:BBBB:BBBB:BBBB::BBBB
```

### PTR records in DNS

Address-to-name lookups in DNS are handled using PTR records. The PTR records for both addresses should be configured to return the official name:

```
address class type name
AA.AA.AA.AA.in-addr.arpa IN PTR host.domain
BB.BB.BB.BB.in-addr.arpa IN PTR host.domain
```

For a dual-stack host:

```
address class type name
AA.AA.AA.AA.in-addr.arpa IN PTR host.domain
BBBB:BBBB:BBBB:BBBB:BBBB:BBBB::BBBB.in-addr.arpa IN PTR host.domain
```

If it is not possible to change the system host name database, create the `hosts` file local to the LSF system, and configure entries for the multi-homed hosts only. Host names and addresses not found in the `hosts` file are looked up in the standard name system on your host.

## Using IPv6 Addresses

IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`; you do not have to map IPv4 addresses to an IPv6 format.

LSF supports IPv6 addresses for the following platforms:

- ◆ Linux 2.4
- ◆ Linux 2.6
- ◆ Solaris 10
- ◆ Windows
  - ◆ XP
  - ◆ 2003
  - ◆ 2000 with Service Pack 1 or higher
- ◆ AIX 5
- ◆ HP-UX
  - ◆ 11i
  - ◆ 11iv1
  - ◆ 11iv2
  - ◆ 11.11
- ◆ SGI Altix ProPack 3, 4, and 5
- ◆ IRIX 6.5.19 and higher, Trusted IRIX 6.5.19 and higher
- ◆ Mac OS 10.2 and higher
- ◆ Cray XT3
- ◆ IBM Power 5 Series

### Enable both IPv4 and IPv6 support

- 
- 1 Configure the parameter `LSF_ENABLE_SUPPORT_IPV6=Y` in `lsf.conf`.
- 

### Configure hosts for IPv6

Follow the steps in this procedure if you do not have an IPv6-enabled DNS server or an IPv6-enabled router. IPv6 is supported on some linux2.4 kernels and on all linux2.6 kernels.

- 
- 1 Configure the kernel.
    - a Does the entry `/proc/net/if_inet6` exist?

- ◆ If *yes*, the kernel is already configured for IPv6. Go to step 2.
  - ◆ If *no*, go to step b.
  - b To load the IPv6 module into the kernel, execute the following command as root:
 

```
modprobe ipv6
```
  - c To check that the module loaded correctly, execute the command
 

```
lsmod | grep -w 'ipv6'
```
  - 2 Add an IPv6 address to the host by executing the following command as root:
 

```
/sbin/ifconfig eth0 inet6 add 3ffe:ffff:0:f101::2/64
```
  - 3 Display the IPv6 address using `ifconfig`.
  - 4 Repeat step 1 through step 3 for other hosts in the cluster.
  - 5 To configure IPv6 networking, add the addresses for all IPv6 hosts to `/etc/hosts` on each host.
- 
- NOTE:** For IPv6 networking, hosts must be on the same subnet.
- 
- 6 Test IPv6 communication between hosts using the command `ping6`.

## Host Groups

You can define a host group within LSF or use an external executable to retrieve host group members.

Use `bhosts` to view a list of existing hosts. Use `bmgroup` to view host group membership use.

### Where to use host groups

LSF host groups can be used in defining the following parameters in LSF configuration files:

- ◆ `HOSTS` in `lsb.queues` for authorized hosts for the queue
- ◆ `HOSTS` in `lsb.hosts` in the `HostPartition` section to list host groups that are members of the host partition

### Configure host groups

- 1 Log in as the LSF administrator to any host in the cluster.
- 2 Open `lsb.hosts`.
- 3 Add the `HostGroup` section if it does not exist.

```
Begin HostGroup
GROUP_NAME GROUP_MEMBER
groupA (all)
groupB (groupA ~hostA ~hostB)
groupC (hostX hostY hostZ)
groupD (groupC ~hostX)
groupE (all ~groupC ~hostB)
groupF (hostF groupC hostK)
```

```
desk_tops (hostD hostE hostF hostG)
Big_servers (!)
End HostGroup
```

- 4 Enter a group name under the GROUP\_NAME column.  
External host groups must be defined in the `egroup` executable.
- 5 Specify hosts in the GROUP\_MEMBER column.  
(Optional) To tell LSF that the group members should be retrieved using `egroup`, put an exclamation mark (!) in the GROUP\_MEMBER column.
- 6 Save your changes.
- 7 Run `badadmin ckconfig` to check the group definition. If any errors are reported, fix the problem and check the configuration again.
- 8 Do one of the following:
  - a Run `badadmin reconfig` if you do not want the new group to be recognized by jobs that were submitted before you reconfigured.
  - b Run `badadmin mbdrestart` if you want the new host to be recognized by jobs that were submitted before you reconfigured.

## Using wildcards and special characters to define host names

You can use special characters when defining host group members under the GROUP\_MEMBER column to specify hosts. These are useful to define several hosts in a single entry, such as for a range of hosts, or for all host names with a certain text string.

If a host matches more than one host group, that host is a member of all groups. If any host group is a condensed host group, the status and other details of the hosts are counted towards all of the matching host groups.

When defining host group members, you can use string literals and the following special characters:

- ◆ Use a tilde (~) to exclude specified hosts or host groups from the list. The tilde can be used in conjunction with the other special characters listed below. The following example matches all hosts in the cluster except for `hostA`, `hostB`, and all members of the `groupA` host group:  

```
... (all ~hostA ~hostB ~groupA)
```
- ◆ Use an asterisk (\*) as a wildcard character to represent any number of characters. The following example matches all hosts beginning with the text string "hostC" (such as `hostCa`, `hostC1`, or `hostCZ1`):  

```
... (hostC*)
```
- ◆ Use square brackets with a hyphen ([*integer1* - *integer2*]) to define a range of non-negative integers at the end of a host name. The first integer must be less than the second integer. The following example matches all hosts from `hostD51` to `hostD100`:  

```
... (hostD[51-100])
```

- ◆ Use square brackets with commas (`[integer1 , integer2 ...]`) to define individual non-negative integers at the end of a host name. The following example matches `hostD101`, `hostD123`, and `hostD321`:

```
... (hostD[101,123,321])
```

- ◆ Use square brackets with commas and hyphens (such as `[integer1 - integer2 , integer3 , integer4 - integer5]`) to define different ranges of non-negative integers at the end of a host name. The following example matches all hosts from `hostD1` to `hostD100`, `hostD102`, all hosts from `hostD201` to `hostD300`, and `hostD320`):

```
... (hostD[1-100,102,201-300,320])
```

## Restrictions

You cannot use more than one set of square brackets in a single host group definition.

The following example is *not* correct:

```
... (hostA[1-10]B[1-20] hostC[101-120])
```

The following example is correct:

```
... (hostA[1-20] hostC[101-120])
```

You cannot define subgroups that contain wildcards and special characters. The following definition for `groupB` is not correct because `groupA` defines hosts with a wildcard:

```
Begin HostGroup
GROUP_NAME GROUP_MEMBER
groupA (hostA*)
groupB (groupA)
End HostGroup
```

## Defining condensed host groups

You can define condensed host groups to display information for its hosts as a summary for the entire group. This is useful because it allows you to see the total statistics of the host group as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and host groups in your cluster.

To define condensed host groups, add a `CONDENSE` column to the `HostGroup` section. Under this column, enter `Y` to define a condensed host group or `N` to define an uncondensed host group, as shown in the following:

```
Begin HostGroup
GROUP_NAME CONDENSE GROUP_MEMBER
groupA Y (hostA hostB hostD)
groupB N (hostC hostE)
End HostGroup
```

The following commands display condensed host group information:

- ◆ `bhosts`
- ◆ `bhosts -w`
- ◆ `bjobs`
- ◆ `bjobs -w`

For the `bhosts` output of this configuration, see [Viewing Host Information](#) on page 56.

## Tuning CPU Factors

Use `bmgroup -l` to see whether host groups are condensed or not.

### Hosts belonging to multiple condensed host groups

If you configure a host to belong to more than one condensed host group using wildcards, `bjobs` can display any of the host groups as execution host name.

For example, host groups `hg1` and `hg2` include the same hosts:

```
Begin HostGroup
GROUP_NAME CONDENSE GROUP_MEMBER # Key words
hg1 Y (host*)
hg2 Y (hos*)
End HostGroup
```

Submit jobs using `bsub -m`:

```
bsub -m "hg2" sleep 1001
```

`bjobs` displays `hg1` as the execution host instead of `hg2`:

```
bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
520 user1 RUN normal host5 hg1 sleep 1001 Apr 15 13:50
521 user1 RUN normal host5 hg1 sleep 1001 Apr 15 13:50
522 user1 PEND normal host5 sleep 1001 Apr 15 13:51
```

### External host group requirements (egroup)

An external host group is a host group for which membership is not statically configured, but is instead retrieved by running an external executable with the name `egroup`. The `egroup` executable must be in the directory specified by `LSF_SERVERDIR`.

This feature allows a site to maintain group definitions outside LSF and import them into LSF configuration at initialization time.

The `egroup` executable is an executable you create yourself that lists group names and hosts that belong to the group.

This executable must have the name `egroup`. When `mbatchd` is restarted, it invokes the `egroup` executable and retrieves groups and group members. The external executable `egroup` runs under the same account as `mbatchd`.

The `egroup` executable must write host names for the host groups to its standard output, each name separated by white space.

The `egroup` executable must recognize the following command, since `mbatchd` invokes external host groups with this command:

```
egroup -m host_group_name
```

where *host\_group\_name* is the name of the host group defined in the executable `egroup` along with its members, and the host group is specified in `lsb.hosts`.

## Tuning CPU Factors

CPU factors are used to differentiate the relative speed of different machines. LSF runs jobs on the best possible machines so that response time is minimized.

To achieve this, it is important that you define correct CPU factors for each machine model in your cluster.

## How CPU factors affect performance

Incorrect CPU factors can reduce performance the following ways.

- ◆ If the CPU factor for a host is too low, that host may not be selected for job placement when a slower host is available. This means that jobs would not always run on the fastest available host.
- ◆ If the CPU factor is too high, jobs are run on the fast host even when they would finish sooner on a slower but lightly loaded host. This causes the faster host to be overused while the slower hosts are underused.

Both of these conditions are somewhat self-correcting. If the CPU factor for a host is too high, jobs are sent to that host until the CPU load threshold is reached. LSF then marks that host as busy, and no further jobs will be sent there. If the CPU factor is too low, jobs may be sent to slower hosts. This increases the load on the slower hosts, making LSF more likely to schedule future jobs on the faster host.

## Guidelines for setting CPU factors

CPU factors should be set based on a benchmark that reflects your workload. If there is no such benchmark, CPU factors can be set based on raw CPU power.

The CPU factor of the slowest hosts should be set to 1, and faster hosts should be proportional to the slowest.

### Example

Consider a cluster with two hosts: `hostA` and `hostB`. In this cluster, `hostA` takes 30 seconds to run a benchmark and `hostB` takes 15 seconds to run the same test. The CPU factor for `hostA` should be 1, and the CPU factor of `hostB` should be 2 because it is twice as fast as `hostA`.

## View normalized ratings

- 1 Run `lsload -N` to display normalized ratings.

LSF uses a normalized CPU performance rating to decide which host has the most available CPU power. Hosts in your cluster are displayed in order from best to worst. Normalized CPU run queue length values are based on an estimate of the time it would take each host to run one additional unit of work, given that an unloaded host with CPU factor 1 runs one unit of work in one unit of time.

## Tune CPU factors

- 1 Log in as the LSF administrator on any host in the cluster.
- 2 Edit `lsf.shared`, and change the `HostModel` section:

```
Begin HostModel
MODELNAME CPUFACTOR ARCHITECTURE # keyword
#HPUX (HPPA)
HP9K712S 2.5 (HP9000712_60)
HP9K712M 2.5 (HP9000712_80)
HP9K712F 4.0 (HP9000712_100)
```

See the *Platform LSF Configuration Reference* for information about the `lsf.shared` file.

- 3 Save the changes to `lsf.shared`.
  - 4 Run `lsadmin reconfig` to reconfigure LIM.
  - 5 Run `badmin reconfig` to reconfigure `mbatchd`.
- 

## Handling Host-level Job Exceptions

You can configure hosts so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

### Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such “black hole”, or “job-eating” hosts exit abnormally. LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (`EXIT_RATE` in `lsb.hosts`).

If `EXIT_RATE` is not specified for the host, LSF invokes `eadmin` if the job exit rate for a host remains above the configured threshold for longer than 10 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change how frequently LSF checks the job exit rate.

Use `GLOBAL_EXIT_RATE` in `lsb.params` to set a cluster-wide threshold in minutes for exited jobs. If `EXIT_RATE` is not specified for the host in `lsb.hosts`, `GLOBAL_EXIT_RATE` defines a default exit rate for all hosts in the cluster. Host-level `EXIT_RATE` overrides the `GLOBAL_EXIT_RATE` value.

### Configuring host exception handling (`lsb.hosts`)

**EXIT\_RATE** Specify a threshold for exited jobs. If the job exit rate is exceeded for 10 minutes or the period specified by `JOB_EXIT_RATE_DURATION` in `lsb.params`, LSF invokes `eadmin` to trigger a host exception.

**Example** The following Host section defines a job exit rate of 20 jobs for all hosts, and an exit rate of 10 jobs on `hostA`.

```
Begin Host
HOST_NAME MXJ EXIT_RATE # Keywords
Default ! 20
hostA ! 10
End Host
```

### Configuring thresholds for host exception handling

By default, LSF checks the number of exited jobs every 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change this default.

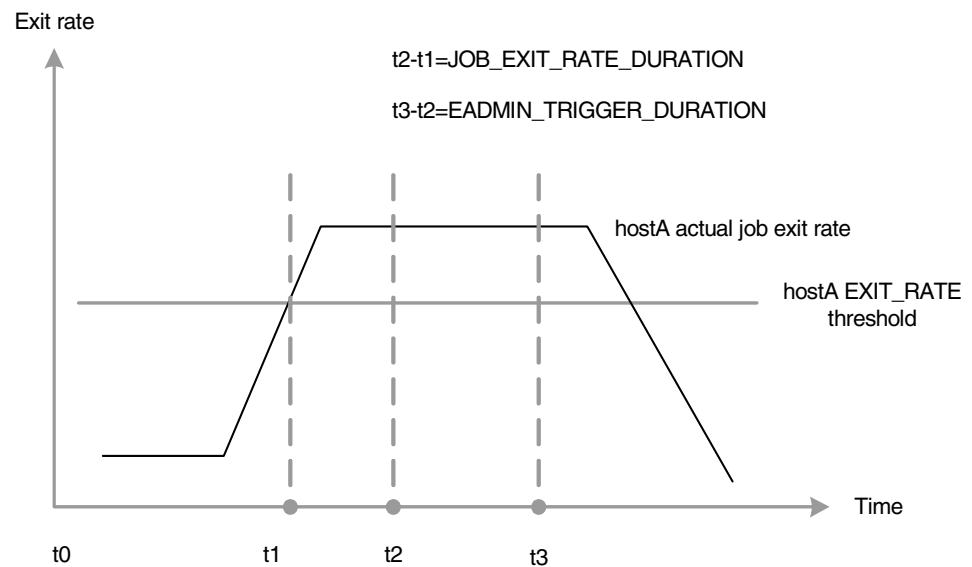


## Tuning

**TIP:** Tune `JOB_EXIT_RATE_DURATION` carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

## Example

In the following diagram, the job exit rate of `hostA` exceeds the configured threshold (`EXIT_RATE` for `hostA` in `lsb.hosts`) LSF monitors `hostA` from time `t1` to time `t2` ( $t2=t1 + \text{JOB\_EXIT\_RATE\_DURATION}$  in `lsb.params`). At `t2`, the exit rate is still high, and a host exception is detected. At `t3` (`EADMIN_TRIGGER_DURATION` in `lsb.params`), LSF invokes `eadmin` and the host exception is handled. By default, LSF closes `hostA` and sends email to the LSF administrator. Since `hostA` is closed and cannot accept any new jobs, the exit rate drops quickly.





## Working with Queues

### Contents

- ◆ [Queue States](#) on page 91
- ◆ [Viewing Queue Information](#) on page 92
- ◆ [Control Queues](#) on page 94
- ◆ [Add and Remove Queues](#) on page 97
- ◆ [Manage Queues](#) on page 99
- ◆ [Handling Job Exceptions in Queues](#) on page 100

### Queue States

Queue states, displayed by `bqueues`, describe the ability of a queue to accept and start batch jobs using a combination of the following states:

- ◆ Open: queues accept new jobs
- ◆ Closed: queues do not accept new jobs
- ◆ Active: queues start jobs on available hosts
- ◆ Inactive: queues hold all jobs

| State         | Description                                                              |
|---------------|--------------------------------------------------------------------------|
| Open:Active   | Accepts and starts new jobs—normal processing                            |
| Open:Inact    | Accepts and holds new jobs—collecting                                    |
| Closed:Active | Does not accept new jobs, but continues to start jobs—draining           |
| Closed:Inact  | Does not accept new jobs and does not start jobs—all activity is stopped |

Queue state can be changed by an LSF administrator or `root`.

Queues can also be activated and inactivated by run windows and dispatch windows (configured in `lsb.queues`, displayed by `bqueues -l`).

`bqueues -l` displays `Inact_Adm` when explicitly inactivated by an Administrator (`badmin qinact`), and `Inact_Win` when inactivated by a run or dispatch window.

## Viewing Queue Information

The `bqueues` command displays information about queues. The `bqueues -l` option also gives current statistics about the jobs in a particular queue, such as the total number of jobs in the queue, the number of jobs running, suspended, and so on.

| To view the...                  | Run...                            |
|---------------------------------|-----------------------------------|
| Available queues                | <code>bqueues</code>              |
| Queue status                    | <code>bqueues</code>              |
| Detailed queue information      | <code>bqueues -l</code>           |
| State change history of a queue | <code>badmin qhist</code>         |
| Queue administrators            | <code>bqueues -l for queue</code> |

In addition to the procedures listed here, see the `bqueues(1)` man page for more details.

### View available queues and queue status

- 1 Run `bqueues`. You can view the current status of a particular queue or all queues. The `bqueues` command also displays available queues in the cluster.

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
interactive 400 Open:Active - - - - 2 0 2 0
priority 43 Open:Active - - - - 16 4 11 1
night 40 Open:Inactive - - - - 4 4 0 0
short 35 Open:Active - - - - 6 1 5 0
license 33 Open:Active - - - - 0 0 0 0
normal 30 Open:Active - - - - 0 0 0 0
idle 20 Open:Active - - - - 6 3 1 2
```

A dash (-) in any entry means that the column does not apply to the row. In this example some queues have no per-queue, per-user or per-processor job limits configured, so the `MAX`, `JL/U` and `JL/P` entries are shown as a dash.

### Job slots required by parallel jobs

**IMPORTANT:** A parallel job with *N* components would require *N* job slots.

### View detailed queue information

- 1 To see the complete status and configuration for each queue, run `bqueues -l`. Specify queue names to select specific queues. The following example displays details for the queue `normal`.

```
bqueues -l normal
QUEUE: normal
--For normal low priority jobs, running only if hosts are lightly loaded. This is
the default queue.
PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P NJOBS PEND RUN SSUSP USUSP
40 20 Open:Active 100 50 11 1 1 0 0 0
Migration threshold is 30 min.
```

```

CPULIMIT RUNLIMIT
20 min of IBM350 342800 min of IBM350

FILELIMIT DATALIMIT STACKLIMIT CORELIMIT MEMLIMIT PROCLIMIT
20000 K 20000 K 2048 K 20000 K 5000 K 3

SCHEDULING PARAMETERS
 r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - 0.7 1.0 0.2 4.0 50 - - - - -
loadStop - 1.5 2.5 - 8.0 240 - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

SCHEDULING POLICIES: FAIRSHARE PREEMPTIVE PREEMPTABLE EXCLUSIVE
USER_SHARES: [groupA, 70] [groupB, 15] [default, 1]

DEFAULT HOST SPECIFICATION : IBM350

RUN_WINDOWS: 2:40-23:00 23:30-1:30
DISPATCH_WINDOWS: 1:00-23:50

USERS: groupA/ groupB/ user5
HOSTS: hostA, hostD, hostB
ADMINISTRATORS: user7
PRE_EXEC: /tmp/apex_pre.x > /tmp/preexec.log 2>&1
POST_EXEC: /tmp/apex_post.x > /tmp/postexec.log 2>&1
REQUEUE_EXIT_VALUES: 45

```

---

## View the state change history of a queue

- 1 Run `badadmin qhist` to display the times when queues are opened, closed, activated, and inactivated.

```
badadmin qhist
```

```
Wed Mar 31 09:03:14: Queue <normal> closed by user or
administrator <root>.
```

```
Wed Mar 31 09:03:29: Queue <normal> opened by user or
administrator <root>.
```

---

## View queue administrators

- 1 Run `bqueues -l` for the queue.

## View exception status for queues (bqueues)

- 1 Use `bqueues` to display the configured threshold for job exceptions and the current number of jobs in the queue in each exception state.  
For example, queue `normal` configures `JOB_IDLE` threshold of 0.10, `JOB_OVERRUN` threshold of 5 minutes, and `JOB_UNDERRUN` threshold of 2 minutes. The following `bqueues` command shows no overrun jobs, one job that finished in less than 2 minutes (underrun) and one job that triggered an idle exception (less than idle factor of 0.10):

```
bqueues -l normal

QUEUE: normal
 -- For normal low priority jobs, running only if hosts are lightly loaded. This
 is the default queue.

PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
 30 20 Open:Active - - - - 0 0 0 0 0 0

STACKLIMIT MEMLIMIT
 2048 K 5000 K

SCHEDULING PARAMETERS
 r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

JOB EXCEPTION PARAMETERS
 OVERRUN(min) UNDERRUN(min) IDLE(cputime/runtime)
Threshold 5 2 0.10
Jobs 0 1 1

USERS: all users
HOSTS: all allremote
CHUNK_JOB_SIZE: 3
```

## Control Queues

Queues are controlled by an LSF Administrator or root issuing a command or through configured dispatch and run windows.

## Close a queue

---

```
1 Run badmin qclose:
 badmin qclose normal
 Queue <normal> is closed
```

When a user tries to submit a job to a closed queue the following message is displayed:

```
bsub -q normal ...
normal: Queue has been closed
```

---

## Open a queue

---

```
1 Run badmin qopen:
 badmin qopen normal
 Queue <normal> is opened
```

---

## Inactivate a queue

---

```
1 Run badmin qinact:
 badmin qinact normal
 Queue <normal> is inactivated
```

---

## Activate a queue

---

```
1 Run badmin qact:
 badmin qact normal
 Queue <normal> is activated
```

---

## Log a comment when controlling a queue

---

```
1 Use the -C option of badmin queue commands qclose, qopen, qact, and
 qinact to log an administrator comment in lsb.events.
```

```
badmin qclose -C "change configuration" normal
```

The comment text `change configuration` is recorded in `lsb.events`.

A new event record is recorded for each queue event. For example:

```
badmin qclose -C "add user" normal
```

followed by

```
badmin qclose -C "add user user1" normal
```

will generate records in `lsb.events`:

```
"QUEUE_CTRL" "7.0 1050082373 1 "normal" 32185 "lsfadmin" "add user"
```

```
"QUEUE_CTRL" "7.0 1050082380 1 "normal" 32185 "lsfadmin" "add user user1"
```

## Control Queues

- 2 Use `badadmin hist` or `badadmin qhist` to display administrator comments for closing and opening hosts.

`badadmin qhist`

```
Fri Apr 4 10:50:36: Queue <normal> closed by administrator
<lsfadmin> change configuration.
```

`bqueues -l` also displays the comment text:

```
bqueues -l normal
```

```
QUEUE: normal
```

```
-- For normal low priority jobs, running only if hosts are lightly loaded. This is the default queue.
```

```
PARAMETERS/STATISTICS
```

```
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
30 20 Closed:Active - - - - 0 0 0 0 0
Interval for a host to accept two jobs is 0 seconds
```

```
THREADLIMIT
```

```
7
```

```
SCHEDULING PARAMETERS
```

```
 r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -
```

```
 cpuspeed bandwidth
loadSched - -
loadStop - -
```

```
JOB EXCEPTION PARAMETERS
```

```
 OVERRUN(min) UNDERRUN(min) IDLE(cputime/runtime)
Threshold - 2 -
Jobs - 0 -
```

```
USERS: all users
```

```
HOSTS: all
```

```
RES_REQ: select[type==any]
```

```
ADMIN ACTION COMMENT: "change configuration"
```

---

## Configure Dispatch Windows

A dispatch window specifies one or more time periods during which batch jobs are dispatched to run on hosts. Jobs are not dispatched outside of configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, queues are always Active; you must explicitly configure dispatch windows in the queue to specify a time when the queue is Inactive.



To configure a dispatch window:

- 1 **Edit** `lsb.queues`
- 2 **Create a DISPATCH\_WINDOW keyword for the queue and specify one or more time windows.**

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY = 45
DISPATCH_WINDOW = 4:30-12:00
End Queue
```
- 3 **Reconfigure the cluster:**
  - a **Run** `lsadmin reconfig.`
  - b **Run** `badadmin reconfig.`
- 4 **Run** `bqueues -l` to display the dispatch windows.

## Configure Run Windows

A run window specifies one or more time periods during which jobs dispatched from a queue are allowed to run. When a run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again. By default, queues are always Active and jobs can run until completion. You must explicitly configure run windows in the queue to specify a time when the queue is Inactive.

To configure a run window:

- 1 **Edit** `lsb.queues.`
- 2 **Create a RUN\_WINDOW keyword for the queue and specify one or more time windows.**

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY = 45
RUN_WINDOW = 4:30-12:00
End Queue
```
- 3 **Reconfigure the cluster:**
  - a **Run** `lsadmin reconfig.`
  - b **Run** `badadmin reconfig.`
- 4 **Run** `bqueues -l` to display the run windows.

## Add and Remove Queues

### Add a queue

- 1 **Log in as the LSF administrator on any host in the cluster.**
- 2 **Edit** `lsb.queues` to add the new queue definition.

## Add and Remove Queues

You can copy another queue definition from this file as a starting point; remember to change the `QUEUE_NAME` of the copied queue.

- 3 Save the changes to `lsb.queues`.
  - 4 Run `badmin reconfig` to reconfigure `mbatchd`.  
Adding a queue does not affect pending or running jobs.
- 

## Remove a queue

**IMPORTANT:** Before removing a queue, make sure there are no jobs in that queue.

If there are jobs in the queue, move pending and running jobs to another queue, then remove the queue. If you remove a queue that has jobs in it, the jobs are temporarily moved to a queue named `lost_and_found`. Jobs in the `lost_and_found` queue remain pending until the user or the LSF administrator uses the `bswitch` command to switch the jobs into regular queues. Jobs in other queues are not affected.

- 1 Log in as the LSF administrator on any host in the cluster.
  - 2 Close the queue to prevent any new jobs from being submitted.  

```
badmin qclose night
Queue <night> is closed
```
  - 3 Move all pending and running jobs into another queue.  
Below, the `bswitch -q night` argument chooses jobs from the `night` queue, and the job ID number `0` specifies that all jobs should be switched:  

```
bjobs -u all -q night
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBM
IT_TIME
5308 user5 RUN night hostA hostD job5 Nov 2
1 18:16
5310 user5 PEND night hostA hostC job10 Nov 2
1 18:17
```

```
bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```
  - 4 Edit `lsb.queues` and remove or comment out the definition for the queue being removed.
  - 5 Save the changes to `lsb.queues`.
  - 6 Run `badmin reconfig` to reconfigure `mbatchd`.
-

## Manage Queues

### Restrict host use by queues

You may want a host to be used only to run jobs submitted to specific queues. For example, if you just added a host for a specific department such as engineering, you may only want jobs submitted to the queues `engineering1` and `engineering2` to be able to run on the host.

- 1 Log on as root or the LSF administrator on any host in the cluster.
- 2 Edit `lsb.queues`, and add the host to the `HOSTS` parameter of specific queues.

```
Begin Queue
QUEUE_NAME = queue1
...
HOSTS=mynewhost hostA hostB
...
End Queue
```

- 3 Save the changes to `lsb.queues`.
- 4 Use `badadmin ckconfig` to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
- 5 Run `badadmin reconfig` to reconfigure `mbatchd`.
- 6 If you add a host to a queue, the new host will not be recognized by jobs that were submitted before you reconfigured. If you want the new host to be recognized, you must use the command `badadmin mbdrestart`.

### Add queue administrators

Queue administrators are optionally configured after installation. They have limited privileges; they can perform administrative operations (open, close, activate, inactivate) on the specified queue, or on jobs running in the specified queue. Queue administrators cannot modify configuration files, or operate on LSF daemons or on queues they are not configured to administer.

To switch a job from one queue to another, you must have administrator privileges for both queues.

- 1 In the `lsb.queues` file, between `Begin Queue` and `End Queue` for the appropriate queue, specify the `ADMINISTRATORS` parameter, followed by the list of administrators for that queue. Separate the administrator names with a space. You can specify user names and group names.

```
Begin Queue
ADMINISTRATORS = User1 GroupA
End Queue
```

## Handling Job Exceptions in Queues

You can configure queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

### Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- ◆ Job underrun—jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally
- ◆ Job overrun—job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for job overrun.
- ◆ Idle job—running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for idle jobs.

### Configuring job exception handling (`lsb.queues`)

You can configure your queues to detect job exceptions. Use the following parameters:

|                     |                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>JOB_IDLE</b>     | Specify a threshold for idle jobs. The value should be a number between 0.0 and 1.0 representing CPU time/runtime. If the job idle factor is less than the specified threshold, LSF invokes <code>eadmin</code> to trigger the action for a job idle exception. |
| <b>JOB_OVERRUN</b>  | Specify a threshold for job overrun. If a job runs longer than the specified run time, LSF invokes <code>eadmin</code> to trigger the action for a job overrun exception.                                                                                       |
| <b>JOB_UNDERRUN</b> | Specify a threshold for job underrun. If a job exits before the specified number of minutes, LSF invokes <code>eadmin</code> to trigger the action for a job underrun exception.                                                                                |

### Example

The following queue defines thresholds for all types job exceptions:

```
Begin Queue
...
JOB_UNDERRUN = 2
JOB_OVERRUN = 5
JOB_IDLE = 0.10
...
End Queue
```

For this queue:

- ◆ A job underrun exception is triggered for jobs running less than 2 minutes
- ◆ A job overrun exception is triggered for jobs running longer than 5 minutes
- ◆ A job idle exception is triggered for jobs with an idle factor (CPU time/runtime) less than 0.10

## Configuring thresholds for job exception handling

By default, LSF checks for job exceptions every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for overrun, underrun, and idle jobs.

### Tuning

**TIP:** Tune `EADMIN_TRIGGER_DURATION` carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.



# Managing Jobs

## Contents

- ◆ [Understanding Job States](#) on page 103
- ◆ [View Job Information](#) on page 106
- ◆ [Changing Job Order Within Queues](#) on page 109
- ◆ [Switch Jobs from One Queue to Another](#) on page 110
- ◆ [Forcing Job Execution](#) on page 111
- ◆ [Suspending and Resuming Jobs](#) on page 111
- ◆ [Killing Jobs](#) on page 112
- ◆ [Sending a Signal to a Job](#) on page 113
- ◆ [Using Job Groups](#) on page 114
- ◆ [Handling Job Exceptions](#) on page 123

## Understanding Job States

The `bjobs` command displays the current state of the job.

### Normal job states

Most jobs enter only three states:

| Job state | Description                                    |
|-----------|------------------------------------------------|
| PEND      | Waiting in a queue for scheduling and dispatch |
| RUN       | Dispatched to a host and running               |
| DONE      | Finished normally with a zero exit value       |

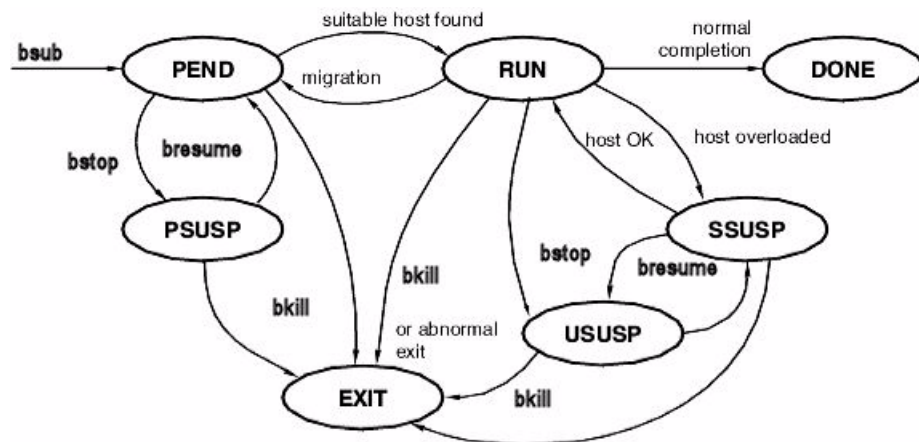
### Suspended job states

If a job is suspended, it has three states:

| Job state | Description                                                            |
|-----------|------------------------------------------------------------------------|
| PSUSP     | Suspended by its owner or the LSF administrator while in PEND state    |
| USUSP     | Suspended by its owner or the LSF administrator after being dispatched |
| SSUSP     | Suspended by the LSF system after being dispatched                     |

## State transitions

A job goes through a series of state transitions until it eventually completes its task, fails, or is terminated. The possible states of a job during its life cycle are shown in the diagram.



## Pending jobs

A job remains pending until all conditions for its execution are met. Some of the conditions are:

- ◆ Start time specified by the user when the job is submitted
- ◆ Load conditions on qualified hosts
- ◆ Dispatch windows during which the queue can dispatch and qualified hosts can accept jobs
- ◆ Run windows during which jobs from the queue can run
- ◆ Limits on the number of job slots configured for a queue, a host, or a user
- ◆ Relative priority to other users and jobs
- ◆ Availability of the specified resources
- ◆ Job dependency and pre-execution conditions

### Maximum pending job threshold

If the user or user group submitting the job has reached the pending job threshold as specified by `MAX_PEND_JOBS` (either in the `User` section of `lsb.users`, or cluster-wide in `lsb.params`), LSF will reject any further job submission requests sent by that user or user group. The system will continue to send the job submission requests with the interval specified by `SUB_TRY_INTERVAL` in `lsb.params` until it has made a number of attempts equal to the `LSB_NTRIES` environment variable. If



LSB\_NTRIES is undefined and LSF rejects the job submission request, the system will continue to send the job submission requests indefinitely as the default behavior.

## Suspended jobs

A job can be suspended at any time. A job can be suspended by its owner, by the LSF administrator, by the root user (superuser), or by LSF.

After a job has been dispatched and started on a host, it can be suspended by LSF. When a job is running, LSF periodically checks the load level on the execution host. If any load index is beyond either its per-host or its per-queue suspending conditions, the lowest priority batch job on that host is suspended.

If the load on the execution host or hosts becomes too high, batch jobs could be interfering among themselves or could be interfering with interactive jobs. In either case, some jobs should be suspended to maximize host performance or to guarantee interactive response time.

LSF suspends jobs according to the priority of the job's queue. When a host is busy, LSF suspends lower priority jobs first unless the scheduling policy associated with the job dictates otherwise.

Jobs are also suspended by the system if the job queue has a run window and the current time goes outside the run window.

A system-suspended job can later be resumed by LSF if the load condition on the execution hosts falls low enough or when the closed run window of the queue opens again.

## WAIT state (chunk jobs)

If you have configured chunk job queues, members of a chunk job that are waiting to run are displayed as WAIT by `bjobs`. Any jobs in WAIT status are included in the count of pending jobs by `bqueues` and `busers`, even though the entire chunk job has been dispatched and occupies a job slot. The `bhosts` command shows the single job slot occupied by the entire chunk job in the number of jobs shown in the NJOBS column.

You can switch (`bswitch`) or migrate (`bmig`) a chunk job member in WAIT state to another queue.

See Chapter 29, “[Chunk Job Dispatch](#)” for more information about chunk jobs.

## Exited jobs

A job might terminate abnormally for various reasons. Job termination can happen from any state. An abnormally terminated job goes into EXIT state. The situations where a job terminates abnormally include:

- ◆ The job is cancelled by its owner or the LSF administrator while pending, or after being dispatched to a host.
- ◆ The job is not able to be dispatched before it reaches its termination deadline, and thus is aborted by LSF.
- ◆ The job fails to start successfully. For example, the wrong executable is specified by the user when the job is submitted.

The job exits with a non-zero exit status.

## View Job Information

You can configure hosts so that LSF detects an abnormally high rate of job exit from a host. See [Handling Host-level Job Exceptions](#) on page 88 for more information.

## Post-execution states

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

The DONE or EXIT job states do not indicate whether post-processing is complete, so jobs that depend on processing may start prematurely. Use the `post_done` and `post_err` keywords on the `bsub -w` command to specify job dependency conditions for job post-processing. The corresponding job states POST\_DONE and POST\_ERR indicate the state of the post-processing.

After the job completes, you cannot perform any job control on the post-processing. Post-processing exit codes are not reported to LSF.

See Chapter 35, “[Pre-Execution and Post-Execution Commands](#)” for more information.

## View Job Information

The `bjobs` command is used to display job information. By default, `bjobs` displays information for the user who invoked the command. For more information about `bjobs`, see the *LSF Reference* and the `bjobs(1)` man page.

## View all jobs for all users

- 1 Run `bjobs -u all` to display all jobs for all users.

Job information is displayed in the following order:

- ◆ Running jobs
- ◆ Pending jobs in the order in which they are scheduled
- ◆ Jobs in high-priority queues are listed before those in lower-priority queues

For example:

```
bjobs -u all
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
1004 user1 RUN short hostA hostA job0 Dec 16 09:23
1235 user3 PEND priority hostM job1 Dec 11 13:55
1234 user2 SSUSP normal hostD hostM job3 Dec 11 10:09
1250 user1 PEND short hostA job4 Dec 11 13:59
```

## View jobs for specific users

- 1 Run `bjobs -u user_name` to display jobs for a specific user:

```
bjobs -u user1
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
2225 user1 USUSP normal hostA job1 Nov 16 11:55
```

|      |       |       |        |       |      |              |
|------|-------|-------|--------|-------|------|--------------|
| 2226 | user1 | PSUSP | normal | hostA | job2 | Nov 16 12:30 |
| 2227 | user1 | PSUSP | normal | hostA | job3 | Nov 16 12:31 |

## View running jobs

- 1 Run `bjobs -r` to display running jobs.

## View done jobs

- 1 Run `bjobs -d` to display recently completed jobs.

## View pending job information

- 1 Run `bjobs -p` to display the reason why a job is pending.
- 2 Run `busers -w all` to see the maximum pending job threshold for all users.

## View suspension reasons

- 1 Run `bjobs -s` to display the reason why a job was suspended.

## View chunk job wait status and wait reason

- 1 Run `bhist -l` to display jobs in WAIT status. Jobs are shown as `Waiting ...`.  
The `bjobs -l` command does not display a WAIT reason in the list of pending jobs.

## View post-execution states

- 1 Run `bhist` to display the POST\_DONE and POST\_ERR states.  
The resource usage of post-processing is not included in the job resource usage.

## View exception status for jobs (bjobs)

- 1 Run `bjobs` to display job exceptions. `bjobs -l` shows exception information for unfinished jobs, and `bjobs -x -l` shows finished as well as unfinished jobs.

For example, the following `bjobs` command shows that job 2 is running longer than the configured `JOB_OVERRUN` threshold, and is consuming no CPU time. `bjobs` displays the job idle factor, and both job overrun and job idle exceptions. Job 1 finished before the configured `JOB_UNDERRUN` threshold, so `bjobs` shows exception status of underrun:

```
bjobs -x -l -a
Job <2>, User <user1>, Project <default>, Status <RUN>, Queue <normal>, Command
 <sleep 600>
Wed Aug 13 14:23:35: Submitted from host <hostA>, CWD <${HOME}>, Output File
 </dev/null>, Specified Hosts <hostB>;
Wed Aug 13 14:23:43: Started on <hostB>, Execution Home </home/user1>, Execution
 CWD </home/user1>;
Resource usage collected.
 IDLE_FACTOR(cputime/runtime): 0.00
 MEM: 3 Mbytes; SWAP: 4 Mbytes; NTHREAD: 3
 PGID: 5027; PIDs: 5027 5028 5029

SCHEDULING PARAMETERS:
 r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

EXCEPTION STATUS: overrun idle

Job <1>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Command
 <sleep 20>
Wed Aug 13 14:18:00: Submitted from host <hostA>, CWD <${HOME}>,
 Output File </dev/null>, Specified Hosts <
 hostB>;
Wed Aug 13 14:18:10: Started on <hostB>, Execution Home </home/user1>, Execution
 CWD </home/user1>;
Wed Aug 13 14:18:50: Done successfully. The CPU time used is 0.2 seconds.
```

## SCHEDULING PARAMETERS:

|           | r15s | r1m | r15m | ut | pg | io | ls | it | tmp | swp | mem |
|-----------|------|-----|------|----|----|----|----|----|-----|-----|-----|
| loadSched | -    | -   | -    | -  | -  | -  | -  | -  | -   | -   | -   |
| loadStop  | -    | -   | -    | -  | -  | -  | -  | -  | -   | -   | -   |

|           | cpuspeed | bandwidth |
|-----------|----------|-----------|
| loadSched | -        | -         |
| loadStop  | -        | -         |

EXCEPTION STATUS: underrun

Use `bacct -l -x` to trace the history of job exceptions.

## Changing Job Order Within Queues

By default, LSF dispatches jobs in a queue in the order of arrival (that is, first-come-first-served), subject to availability of suitable server hosts.

Use the `btop` and `bbot` commands to change the position of pending jobs, or of pending job array elements, to affect the order in which jobs are considered for dispatch. Users can only change the relative position of their own jobs, and LSF administrators can change the position of any users' jobs.

### bbot

Moves jobs relative to your last job in the queue.

If invoked by a regular user, `bbot` moves the selected job after the last job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, `bbot` moves the selected job after the last job with the same priority submitted to the queue.

### btop

Moves jobs relative to your first job in the queue.

If invoked by a regular user, `btop` moves the selected job before the first job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, `btop` moves the selected job before the first job with the same priority submitted to the queue.

## Moving a job to the top of the queue

In the following example, job 5311 is moved to the top of the queue. Since job 5308 is already running, job 5311 is placed in the queue after job 5308.

Note that `user1`'s job is still in the same position on the queue. `user2` cannot use `btop` to get extra jobs at the top of the queue; when one of his jobs moves up the queue, the rest of his jobs move down.

```
bjobs -u all
```

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 5308  | user2 | RUN  | normal | hostA     | hostD     | /s500    | Oct 23 10:16 |
| 5309  | user2 | PEND | night  | hostA     |           | /s200    | Oct 23 11:04 |

## Switch Jobs from One Queue to Another

```
5310 user1 PEND night hostB /myjob Oct 23 13:45
5311 user2 PEND night hostA /s700 Oct 23 18:17
```

```
btop 5311
```

```
Job <5311> has been moved to position 1 from top.
```

```
bjobs -u all
```

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 5308  | user2 | RUN  | normal | hostA     | hostD     | /s500    | Oct 23 10:16 |
| 5311  | user2 | PEND | night  | hostA     |           | /s200    | Oct 23 18:17 |
| 5310  | user1 | PEND | night  | hostB     |           | /myjob   | Oct 23 13:45 |
| 5309  | user2 | PEND | night  | hostA     |           | /s700    | Oct 23 11:04 |

## Switch Jobs from One Queue to Another

You can use the command `bswitch` to change jobs from one queue to another. This is useful if you submit a job to the wrong queue, or if the job is suspended because of queue thresholds or run windows and you would like to resume the job.

### Switch a single job to a different queue

- 1 Run `bswitch` to move pending and running jobs from queue to queue.

In the following example, job 5309 is switched to the `priority` queue:

```
bswitch priority 5309
```

```
Job <5309> is switched to queue <priority>
```

```
bjobs -u all
```

| JOBID | USER  | STAT | QUEUE    | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME  |
|-------|-------|------|----------|-----------|-----------|----------|--------------|
| 5308  | user2 | RUN  | normal   | hostA     | hostD     | /job500  | Oct 23 10:16 |
| 5309  | user2 | RUN  | priority | hostA     | hostB     | /job200  | Oct 23 11:04 |
| 5311  | user2 | PEND | night    | hostA     |           | /job700  | Oct 23 18:17 |
| 5310  | user1 | PEND | night    | hostB     |           | /myjob   | Oct 23 13:45 |

### Switch all jobs to a different queue

- 1 Run `bswitch -q from_queue to_queue 0` to switch all the jobs in a queue to another queue.

The `-q` option is used to operate on all jobs in a queue. The job ID number 0 specifies that all jobs from the `night` queue should be switched to the `idle` queue:

The example below selects jobs from the `night` queue and switches them to the `idle` queue.

```
bswitch -q night idle 0
```

```
Job <5308> is switched to queue <idle>
```

```
Job <5310> is switched to queue <idle>
```

## Forcing Job Execution

A pending job can be forced to run with the `brun` command. This operation can only be performed by an LSF administrator.

You can force a job to run on a particular host, to run until completion, and other restrictions. For more information, see the `brun` command.

When a job is forced to run, any other constraints associated with the job such as resource requirements or dependency conditions are ignored.

In this situation you may see some job slot limits, such as the maximum number of jobs that can run on a host, being violated. A job that is forced to run cannot be preempted.

### Force a pending job to run

---

1 Run `brun -m hostname job_ID` to force a pending job to run.

You must specify the host on which the job will run.

For example, the following command will force the sequential job 104 to run on `hostA`:

```
brun -m hostA 104
```

---

## Suspending and Resuming Jobs

A job can be suspended by its owner or the LSF administrator. These jobs are considered user-suspended and are displayed by `bjobs` as `USUSP`.

If a user suspends a high priority job from a non-preemptive queue, the load may become low enough for LSF to start a lower priority job in its place. The load created by the low priority job can prevent the high priority job from resuming. This can be avoided by configuring preemptive queues.

### Suspend a job

---

1 Run `bstop job_ID`.

Your job goes into `USUSP` state if the job is already started, or into `PSUSP` state if it is pending.

```
bstop 3421
Job <3421> is being stopped
```

The above example suspends job 3421.

---

### UNIX

`bstop` sends the following signals to the job:

- ◆ `SIGTSTP` for parallel or interactive jobs—`SIGTSTP` is caught by the master process and passed to all the slave processes running on other hosts.
- ◆ `SIGSTOP` for sequential jobs—`SIGSTOP` cannot be caught by user programs. The `SIGSTOP` signal can be configured with the `LSB_SIGSTOP` parameter in `lsf.conf`.

## Resume a job

`bstop` causes the job to be suspended.

---

```
1 Run bresume job_ID:
bresume 3421
Job <3421> is being resumed
resumes job 3421.
```

Resuming a user-suspended job does not put your job into `RUN` state immediately. If your job was running before the suspension, `bresume` first puts your job into `SSUSP` state and then waits for `sbatchd` to schedule it according to the load conditions.

---

## Killing Jobs

The `bkill` command cancels pending batch jobs and sends signals to running jobs. By default, on UNIX, `bkill` sends the `SIGKILL` signal to running jobs.

Before `SIGKILL` is sent, `SIGINT` and `SIGTERM` are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from `mbatchd` to `sbatchd`. `sbatchd` waits for the job to exit before reporting the status. Because of these delays, for a short period of time after the `bkill` command has been issued, `bjobs` may still report that the job is running.

On Windows, job control messages replace the `SIGINT` and `SIGTERM` signals, and termination is implemented by the `TerminateProcess()` system call.

## Kill a job

---

```
1 Run bkill job_ID. For example, the following command kills job 3421:
bkill 3421
Job <3421> is being terminated
```

---

## Kill multiple jobs

---

```
1 Run bkill 0 to kill all pending jobs in the cluster or use bkill 0 with the -g, -J, -m, -q, or -u options to kill all jobs that satisfy these options.
```

The following command kills all jobs dispatched to the `hostA` host:

```
bkill -m hostA 0
Job <267> is being terminated
Job <268> is being terminated
Job <271> is being terminated
```

The following command kills all jobs in the `groupA` job group:

```
bkill -g groupA 0
Job <2083> is being terminated
```



---

```
Job <2085> is being terminated
```

---

## Kill a large number of jobs rapidly

Killing multiple jobs with `bkill 0` and other commands is usually sufficient for moderate numbers of jobs. However, killing a large number of jobs (approximately greater than 1000 jobs) can take a long time to finish.

- 1 Run `bkill -b` to kill a large number of jobs faster than with normal means. However, jobs killed in this manner are not logged to `lsb.acct`.  
Local pending jobs are killed immediately and cleaned up as soon as possible, ignoring the time interval specified by `CLEAN_PERIOD` in `lsb.params`. Other jobs are killed as soon as possible but cleaned up normally (after the `CLEAN_PERIOD` time interval).  
If the `-b` option is used with `bkill 0`, it kills all applicable jobs and silently skips the jobs that cannot be killed.  
The `-b` option is ignored if used with `-r` or `-s`.
- 

## Force removal of a job from LSF

- 1 Run `bkill -r` to force the removal of the job from LSF. Use this option when a job cannot be killed in the operating system.  
The `bkill -r` command removes a job from the LSF system without waiting for the job to terminate in the operating system. This sends the same series of signals as `bkill` without `-r`, except that the job is removed from the system immediately, the job is marked as `EXIT`, and job resources that LSF monitors are released as soon as LSF receives the first signal.
- 

## Sending a Signal to a Job

LSF uses signals to control jobs, to enforce scheduling policies, or in response to user requests. The principal signals LSF uses are `SIGSTOP` to suspend a job, `SIGCONT` to resume a job, and `SIGKILL` to terminate a job.

Occasionally, you may want to override the default actions. For example, instead of suspending a job, you might want to kill or checkpoint it. You can override the default job control actions by defining the `JOB_CONTROLS` parameter in your queue configuration. Each queue can have its separate job control actions.

You can also send a signal directly to a job. You cannot send arbitrary signals to a pending job; most signals are only valid for running jobs. However, LSF does allow you to kill, suspend and resume pending jobs.

You must be the owner of a job or an LSF administrator to send signals to a job.

You use the `bkill -s` command to send a signal to a job. If you issue `bkill` without the `-s` option, a `SIGKILL` signal is sent to the specified jobs to kill them. Twenty seconds before `SIGKILL` is sent, `SIGTERM` and `SIGINT` are sent to give the job a chance to catch the signals and clean up.

On Windows, job control messages replace the `SIGINT` and `SIGTERM` signals, but only customized applications are able to process them. Termination is implemented by the `TerminateProcess()` system call.

### Signals on different platforms

LSF translates signal numbers across different platforms because different host types may have different signal numbering. The real meaning of a specific signal is interpreted by the machine from which the `bkill` command is issued.

For example, if you send signal 18 from a SunOS 4.x host, it means `SIGTSTP`. If the job is running on HP-UX and `SIGTSTP` is defined as signal number 25, LSF sends signal 25 to the job.

### Send a signal to a job

On most versions of UNIX, signal names and numbers are listed in the `kill(1)` or `signal(2)` man pages. On Windows, only customized applications are able to process job control messages specified with the `-s` option.

---

1 Run `bkill -s signal job_id`, where *signal* is either the signal name or the signal number:

```
bkill -s TSTP 3421
Job <3421> is being signaled
```

The above example sends the `TSTP` signal to job 3421.

---

## Using Job Groups

A collection of jobs can be organized into job groups for easy management. A job group is a container for jobs in much the same way that a directory in a file system is a container for files. For example, a payroll application may have one group of jobs that calculates weekly payments, another job group for calculating monthly salaries, and a third job group that handles the salaries of part-time or contract employees. Users can submit, view, and control jobs according to their groups rather than looking at individual jobs.

### Job group hierarchy

Jobs in job groups are organized into a hierarchical tree similar to the directory structure of a file system. Like a file system, the tree contains groups (which are like directories) and jobs (which are like files). Each group can contain other groups or individual jobs. Job groups are created independently of jobs, and can have dependency conditions which control when jobs within the group are considered for scheduling.

### Job group path

The *job group path* is the name and location of a job group within the job group hierarchy. Multiple levels of job groups can be defined to form a hierarchical tree. A job group can contain jobs and sub-groups.

## Root job group

LSF maintains a single tree under which all jobs in the system are organized. The top-most level of the tree is represented by a top-level “root” job group, named “/”. The root group is owned by the primary LSF Administrator and cannot be removed. Users create new groups under the root group. By default, if you do not specify a job group path name when submitting a job, the job is created under the top-level “root” job group, named “/”.

## Job group owner

Each group is owned by the user who created it. The login name of the user who creates the job group is the job group owner. Users can add job groups into a groups that are owned by other users, and they can submit jobs to groups owned by other users.

## Job control under job groups

Job owners can control their own jobs attached to job groups as usual. Job group owners can also control any job under the groups they own and below.

For example:

- ◆ Job group /A is created by user1
- ◆ Job group /A/B is created by user2
- ◆ Job group /A/B/C is created by user3

All users can submit jobs to any job group, and control the jobs they own in all job groups. For jobs submitted by other users:

- ◆ user1 can control jobs submitted by other users in all 3 job groups: /A, /A/B, and /A/B/C
- ◆ user2 can control jobs submitted by other users only in 2 job groups: /A/B and /A/B/C
- ◆ user3 can control jobs submitted by other users only in job group /A/B/C

The LSF administrator can control jobs in any job group.

## Default job group

You can specify a default job group for jobs submitted without explicitly specifying a job group. LSF associates the job with the job group specified with `DEFAULT_JOBGROUP` in `lsb.params`. The `LSB_DEFAULT_JOBGROUP` environment variable overrides the setting of `DEFAULT_JOBGROUP`. The `bsub -g job_group_name` option overrides both `LSB_DEFAULT_JOBGROUP` and `DEFAULT_JOBGROUP`.

Default job group specification supports macro substitution for project name (`%p`) and user name (`%u`). When you specify `bsub -P project_name`, the value of `%p` is the specified project name. If you do not specify a project name at job submission, `%p` is the project name defined by setting the environment variable `LSB_DEFAULTPROJECT`, or the project name specified by `DEFAULT_PROJECT` in `lsb.params`. the default project name is `default`.

For example, a default job group name specified by `DEFAULT_JOBGROUP=/canada/%p/%u` is expanded to the value for the LSF project name and the user name of the job submission user (for example, `/canada/projects/user1`).

Job group names must follow this format:

- ◆ Job group names must start with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/B/C` is correct, but `DEFAULT_JOBGROUP=A/B/C` is not correct.
- ◆ Job group names cannot end with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/` is not correct.
- ◆ Job group names cannot contain more than one slash character (/) in a row. For example, job group names like `DEFAULT_JOBGROUP=/A//B` or `DEFAULT_JOBGROUP=A///B` are not correct.
- ◆ Job group names cannot contain spaces. For example, `DEFAULT_JOBGROUP=/A/B C/D` is not correct.
- ◆ Project names and user names used for macro substitution with `%p` and `%u` cannot start or end with slash character (/).
- ◆ Project names and user names used for macro substitution with `%p` and `%u` cannot contain spaces or more than one slash character (/) in a row.
- ◆ Project names or user names containing slash character (/) will create separate job groups. For example, if the project name is `canada/projects`, `DEFAULT_JOBGROUP=%p` results in a job group hierarchy `/canada/projects`.

## Job group limits

Job group limits apply to the job group hierarchy. The job group limit is a positive number greater than or equal to zero (0), specifying the maximum the number of running and suspended jobs under the job group (including child groups). If limit is zero (0), no jobs under the job group can run.

By default, a job group has no limit. Limits persist across `mbatchd` restart and reconfiguration.

You cannot specify a limit for the root job group. The root job group has no job limit. Job groups added with no limits specified inherit any limits of existing parent job groups. The `-L` option only limits the lowest level job group created.

The maximum number of running and suspended jobs (including USSUP and SSUP) in a job group cannot exceed the limit defined on the job group and its parent job group.

The job group limit is based on the number of running and suspended jobs in the job group. If you specify a job group limit as 2, at most 2 jobs can run under the group at any time, regardless of how many jobs or job slots are used. If the currently available job slots is zero (0), even if the job group job limit is not exceeded, LSF cannot dispatch a job to the job group.

If a parallel job requests 2 CPUs (`bsub -n 2`), the job group limit is per job, not per slots used by the job.

A job array may also be under a job group, so job arrays also support job group limits.

Job group limits are not supported at job submission for job groups created automatically with `bsub -g`. Use `bgadd -L` before job submission.

Jobs forwarded to the execution cluster in a MultiCluster environment are not counted towards the job group limit.

## Examples

```
bgadd -L 6 /canada/projects/test
```

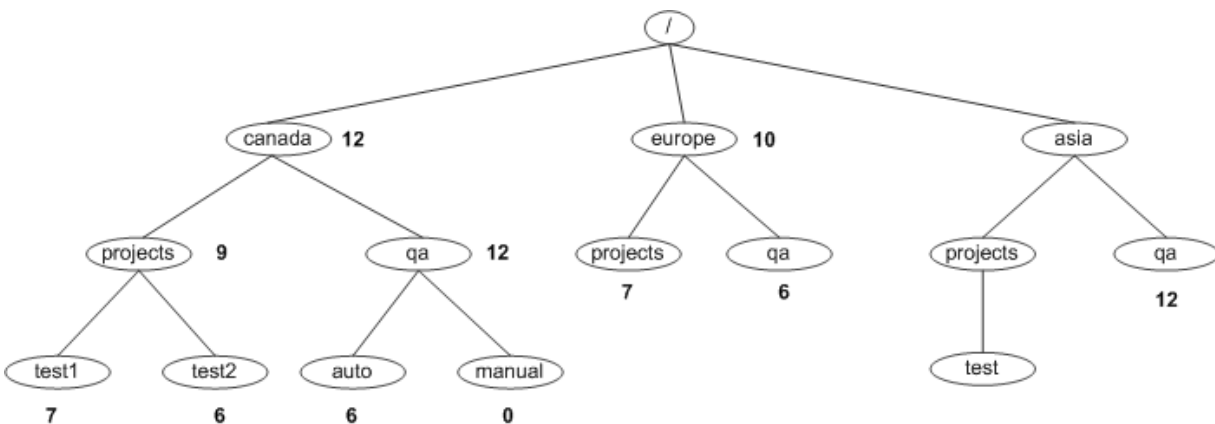
If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit of `/canada` does not change.

The limits on child job groups cannot exceed the parent job group limit. For example, if `/canada/projects` has a limit of 5:

```
bgadd -L 6 /canada/projects/test
```

is rejected because `/canada/projects/test` attempts to increase the limit of its parent `/canada/projects` from 5 to 6.

## Example job group hierarchy with limits



In this configuration:

- ◆ Every node is a job group, including the root (/) job group
- ◆ The root (/) job group cannot have any limit definition
- ◆ By default, child groups have the same limit definition as their direct parent group, so `/asia`, `/asia/projects`, and `/asia/projects/test` all have no limit
- ◆ The number of running and suspended jobs in a job group (including all of its child groups) cannot exceed the defined limit
- ◆ If there are 7 running or suspended jobs in job group `/canada/projects/test1`, even though the job limit of group `/canada/qa/auto` is 6, `/canada/qa/auto` can only have a maximum of 5 running and suspended ( $12-7=5$ )

- ◆ When a job is submitted to a job group, LSF checks the limits for the entire job group. For example, for a job is submitted to job group `/canada/qa/auto`, LSF checks the limits on groups `/canada/qa/auto`, `/canada/qa` and `/canada`. If any one limit in the branch of the hierarchy is exceeded, the job remains pending
- ◆ The zero (0) job limit for job group `/canada/qa/manual` means no job in the job group can enter running status

## Create a job group

- 1 Use the `bgadd` command to create a new job group.

You must provide full group path name for the new job group. The last component of the path is the name of the new group to be created:

```
bgadd /risk_group
```

The above example creates a job group named `risk_group` under the root group `/`.

```
bgadd /risk_group/portfolio1
```

The above example creates a job group named `portfolio1` under job group `/risk_group`.

```
bgadd /risk_group/portfolio1/current
```

The above example creates a job group named `current` under job group `/risk_group/portfolio1`.

If the group hierarchy `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy.

## Submit jobs under a job group

- 1 Use the `-g` option of `bsub` to submit a job into a job group.

The job group does not have to exist before submitting the job.

```
bsub -g /risk_group/portfolio1/current myjob
Job <105> is submitted to default queue.
```

Submits `myjob` to the job group `/risk_group/portfolio1/current`.

If group `/risk_group/portfolio1/current` exists, job 105 is attached to the job group.

If group `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy and the job is attached to group.

### -g and -sla options

**TIP:** Use `-sla` with `-g` to attach all jobs in a job group to a service class and have them scheduled as SLA jobs. Multiple job groups can be created under the same SLA. You can submit additional jobs to the job group without specifying the service class name again.

## MultiCluster

In a MultiCluster job forwarding mode, job groups only apply on the submission cluster, not on the execution cluster. LSF treats the execution cluster as execution engine, and only enforces job group policies at the submission cluster.

Jobs forwarded to the execution cluster in a MultiCluster environment are not counted towards job group limits.

## View jobs in job groups

View job group information, and jobs running in specific job groups.

### View information about job groups (bjgroup)

- 1 Use the `bjgroup` command to see information about jobs in job groups.

```
bjgroup
GROUP_NAME NJOBS PEND RUN SSUSP USUSP FINISH SLA JLIMIT OWNER
/A 0 0 0 0 0 0 () 0/10 user1
/X 0 0 0 0 0 0 () 0/- user2
/A/B 0 0 0 0 0 0 () 0/5 user1
/X/Y 0 0 0 0 0 0 () 0/5 user2
```

- 2 Use `bjgroup -s` to sort job groups by group hierarchy.

For example, for job groups named `/A`, `/A/B`, `/X` and `/X/Y`, `bjgroup -s` displays:

```
bjgroup -s
GROUP_NAME NJOBS PEND RUN SSUSP USUSP FINISH SLA JLIMIT OWNER
/A 0 0 0 0 0 0 () 0/10 user1
/A/B 0 0 0 0 0 0 () 0/5 user1
/X 0 0 0 0 0 0 () 0/- user2
/X/Y 0 0 0 0 0 0 () 0/5 user2
```

- 3 Specify a job group name to show the hierarchy of a single job group:

```
bjgroup -s /X
GROUP_NAME NJOBS PEND RUN SSUSP USUSP FINISH SLA JLIMIT OWNER
/X 25 0 25 0 0 0 puccini 25/100 user1
/X/Y 20 0 20 0 0 0 puccini 20/30 user1
/X/Z 5 0 5 0 0 0 puccini 5/10 user2
```

- 4 Specify a job group name with a trailing slash character (`/`) to show only the root job group:

```
bjgroup -s /X/
GROUP_NAME NJOBS PEND RUN SSUSP USUSP FINISH SLA JLIMIT OWNER
/X 25 0 25 0 0 0 puccini 25/100 user1
```

- 5 Use `bjgroup -N` to display job group information by job slots instead of number of jobs. `NSLOTS`, `PEND`, `RUN`, `SSUSP`, `USUSP`, `RSV` are all counted in slots rather than number of jobs:

```
bjgroup -N
GROUP_NAME NSLOTS PEND RUN SSUSP USUSP RSV SLA OWNER
/X 25 0 25 0 0 0 puccini user1
```

## Using Job Groups

```
/A/B 20 0 20 0 0 0 0 wagner batch
```

-N by itself shows job slot info for all job groups, and can combine with -s to sort the job groups by hierarchy:

```
bjgroup -N -s
```

```
GROUP_NAME NSLOTS PEND RUN SSUSP USUSP RSV SLA OWNER
/A 0 0 0 0 0 0 wagner batch
/A/B 0 0 0 0 0 0 wagner user1
/X 25 0 25 0 0 0 puccini user1
/X/Y 20 0 20 0 0 0 puccini batch
/X/Z 5 0 5 0 0 0 puccini batch
```

## View jobs for a specific job group (bjobs)

- 1 Run `bjobs -g` and specify a job group path to view jobs attached to the specified group.

```
bjobs -g /risk_group
```

```
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
113 user1 PEND normal hostA hostA myjob Jun 17 16:15
111 user2 RUN normal hostA hostA myjob Jun 14 15:13
110 user1 RUN normal hostB hostA myjob Jun 12 05:03
104 user3 RUN normal hostA hostC myjob Jun 11 13:18
```

`bjobs -l` displays the full path to the group to which a job is attached:

```
bjobs -l -g /risk_group
```

```
Job <101>, User <user1>, Project <default>, Job Group
</risk_group>, Status <RUN>, Queue <normal>, Command <myjob>
Tue Jun 17 16:21:49: Submitted from host <hostA>, CWD
</home/user1>
Tue Jun 17 16:22:01: Started on <hostA>;
...
```

## Control jobs in job groups

Suspend and resume jobs in job groups, move jobs to different job groups, terminate jobs in job groups, and delete job groups.

### Suspend jobs (bstop)

- 1 Use the -g option of `bstop` and specify a job group path to suspend jobs in a job group

```
bstop -g /risk_group 106
Job <106> is being stopped
```



- 2 Use job ID 0 (zero) to suspend all jobs in a job group:

```
bstop -g /risk_group/consolidate 0
Job <107> is being stopped
Job <108> is being stopped
Job <109> is being stopped
```

---

## Resume suspended jobs (bresume)

---

- 1 Use the `-g` option of `bresume` and specify a job group path to resume suspended jobs in a job group:

```
bresume -g /risk_group 106
Job <106> is being resumed
```

- 2 Use job ID 0 (zero) to resume all jobs in a job group:

```
bresume -g /risk_group 0
Job <109> is being resumed
Job <110> is being resumed
Job <112> is being resumed
```

---

## Move jobs to a different job group (bmod)

---

- 1 Use the `-g` option of `bmod` and specify a job group path to move a job or a job array from one job group to another.

```
bmod -g /risk_group/portfolio2/monthly 105
moves job 105 to job group /risk_group/portfolio2/monthly.
```

Like `bsub -g`, if the job group does not exist, LSF creates it.

`bmod -g` cannot be combined with other `bmod` options. It can only operate on pending jobs. It cannot operate on running or finished jobs.

You can modify your own job groups and job groups that other users create under your job groups. The LSF administrator can modify job groups of all users.

You cannot move job array elements from one job group to another, only entire job arrays. If any job array elements in a job array are running, you cannot move the job array to another group. A job array can only belong to one job group at a time.

You cannot modify the job group of a job attached to a service class.

`bhist -l` shows job group modification information:

```
bhist -l 105
```

```
Job <105>, User <user1>, Project <default>, Job Group </risk_group>, Command <myjob>
```

```
Wed May 14 15:24:07: Submitted from host <hostA>, to Queue <normal>, CWD
<${HOME}/lsf51/5.1/sparc-sol7-64/bin>;
```

```
Wed May 14 15:24:10: Parameters of Job are changed:
Job group changes to: /risk_group/portfolio2/monthly;
```

## Using Job Groups

```
Wed May 14 15:24:17: Dispatched to <hostA>;
Wed May 14 15:24:17: Starting (Pid 8602);
...
```

---

## Terminate jobs (bkill)

---

- 1 Use the `-g` option of `bkill` and specify a job group path to terminate jobs in a job group.

```
bkill -g /risk_group 106
Job <106> is being terminated
```

- 2 Use job ID 0 (zero) to terminate all jobs in a job group:

```
bkill -g /risk_group 0
Job <1413> is being terminated
Job <1414> is being terminated
Job <1415> is being terminated
Job <1416> is being terminated
```

`bkill` only kills jobs in the job group you specify. It does not kill jobs in lower level job groups in the path. For example, jobs are attached to job groups `/risk_group` and `/risk_group/consolidate`:

```
bsub -g /risk_group myjob
Job <115> is submitted to default queue <normal>.
bsub -g /risk_group/consolidate myjob2
Job <116> is submitted to default queue <normal>.
```

The following `bkill` command only kills jobs in `/risk_group`, not the subgroup `/risk_group/consolidate`:

```
bkill -g /risk_group 0
Job <115> is being terminated
bkill -g /risk_group/consolidate 0
Job <116> is being terminated
```

---

## Delete a job group (bgdel)

---

- 1 Use `bgdel` command to remove a job group. The job group cannot contain any jobs.

```
bgdel /risk_group
Job group /risk_group is deleted.
```

deletes the job group `/risk_group` and all its subgroups.

---

## Add a job group limit (bgadd)

- 1 Run `bgadd -L limit /job_group_name` to specify a job limit for a job group.

Where *limit* is a positive number greater than or equal to zero (0), specifying the maximum the number of running and suspended jobs under the job group (including child groups) If *limit* is zero (0), no jobs under the job group can run.

For example:

```
bgadd -L 6 /canada/projects/test
```

If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit of `/canada` does not change.

## Modify a job group limit (bgmod)

- 1 Run `bgmod` to change a job group limit.

```
bgmod [-L limit | -Ln] /job_group_name
```

`-L limit` changes the limit of `job_group_name` to the specified value. If the job group has parent job groups, the new limit cannot exceed the limits of any higher level job groups. Similarly, if the job group has child job groups, the new value must be greater than any limits on the lower level job groups.

`-Ln` removes the existing job limit for the job group. If the job group has parent job groups, the job modified group automatically inherits any limits from its direct parent job group.

You must provide full group path name for the modified job group. The last component of the path is the name of the job group to be modified.

Only root, LSF administrators, or the job group creator, or the creator of the parent job groups can use `bgmod` to modify a job group limit.

The following command only modifies the limit of group `/canada/projects/test1`. It does not modify limits of `/canada` or `/canada/projects`.

```
bgmod -L 6 /canada/projects/test1
```

To modify limits of `/canada` or `/canada/projects`, you must specify the exact group name:

```
bgmod -L 6 /canada
```

or

```
bgmod -L 6 /canada/projects
```

## Handling Job Exceptions

You can configure hosts and queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

Run `bjobs -d -m host_name` to see exited jobs for a particular host.

### Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- ◆ Job underrun— jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally

- ◆ Job overrun—job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for job overrun.
- ◆ Idle job—running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for idle jobs.

### Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such “black hole”, or “job-eating” hosts exit abnormally. By default, LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (`EXIT_RATE` in `lsb.hosts`).

If `EXIT_RATE` is not specified for the host, LSF invokes `eadmin` if the job exit rate for a host remains above the configured threshold for longer than 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change how frequently LSF checks the job exit rate.

Use `GLOBAL_EXIT_RATE` in `lsb.params` to set a cluster-wide threshold in minutes for exited jobs. If `EXIT_RATE` is not specified for the host in `lsb.hosts`, `GLOBAL_EXIT_RATE` defines a default exit rate for all hosts in the cluster. Host-level `EXIT_RATE` overrides the `GLOBAL_EXIT_RATE` value.

Jobs killed with `bkill` are counted in the job exit rate.

### Customize job exception actions with the `eadmin` script

When an exception is detected, LSF takes appropriate action by running the script `LSF_SERVERDIR/eadmin` on the master host.

You can customize `eadmin` to suit the requirements of your site. For example, `eadmin` could find out the owner of the problem jobs and use `bstop -u` to stop all jobs that belong to the user.

In some environments, a job running 1 hour would be an overrun job, while this may be a normal job in other environments. If your configuration considers jobs running longer than 1 hour to be overrun jobs, you may want to close the queue when LSF detects a job that has run longer than 1 hour and invokes `eadmin`.

### Default `eadmin` actions

For host-level exceptions, LSF closes the host and sends email to the LSF administrator. The email contains the host name, job exit rate for the host, and other host information. The message `eadmin: JOB EXIT THRESHOLD EXCEEDED` is attached to the closed host event in `lsb.events`, and displayed by `badmin hist` and `badmin hhist`. Only one email is sent for host exceptions.

For job exceptions, LSF sends email to the LSF administrator. The email contains the job ID, exception type (overrun, underrun, idle job), and other job information.

An email is sent for all detected job exceptions according to the frequency configured by `EADMIN_TRIGGER_DURATION` in `lsb.params`. For example, if `EADMIN_TRIGGER_DURATION` is set to 5 minutes, and 1 overrun job and 2 idle jobs are detected, after 5 minutes, `eadmin` is invoked and only one email is sent. If another overrun job is detected in the next 5 minutes, another email is sent.

## Handling job initialization failures

By default, LSF handles job exceptions for jobs that exit after they have started running. You can also configure LSF to handle jobs that exit during initialization because of an execution environment problem, or because of a user action or LSF policy.

LSF detects that the jobs are exiting before they actually start running, and takes appropriate action when the job exit rate exceeds the threshold for specific hosts (`EXIT_RATE` in `lsb.hosts`) or for all hosts (`GLOBAL_EXIT_RATE` in `lsb.params`).

Use `EXIT_RATE_TYPE` in `lsb.params` to include job initialization failures in the exit rate calculation. The following table summarizes the exit rate types you can configure:

| Exit rate type ...                                                                   | Includes ...                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JOBEXIT                                                                              | Local exited jobs<br>Remote job initialization failures<br>Parallel job initialization failures on hosts other than the first execution host<br>Jobs exited by user action (e.g., <code>bkill</code> , <code>bstop</code> , etc.) or LSF policy (e.g., load threshold exceeded, job control action, advance reservation expired, etc.) |
| JOBEXIT_NONLSF<br>This is the default when<br><code>EXIT_RATE_TYPE</code> is not set | Local exited jobs<br>Remote job initialization failures<br>Parallel job initialization failures on hosts other than the first execution host                                                                                                                                                                                           |
| JOBINIT                                                                              | Local job initialization failures<br>Parallel job initialization failures on the first execution host                                                                                                                                                                                                                                  |
| HPCINIT                                                                              | Job initialization failures for Platform LSF HPC jobs                                                                                                                                                                                                                                                                                  |

### Job exits excluded from exit rate calculation

By default, jobs that are exited for non-host related reasons (user actions and LSF policies) are not counted in the exit rate calculation. Only jobs that are exited for what LSF considers host-related problems and are used to calculate a host exit rate.

The following cases are *not included* in the exit rate calculations:

- ◆ `bkill`, `bkill -r`
- ◆ `brequeue`
- ◆ RERUNNABLE jobs killed when a host is unavailable
- ◆ Resource usage limit exceeded (for example, `PROCESSLIMIT`, `CPULIMIT`, etc.)

- ◆ Queue-level job control action TERMINATE and TERMINATE\_WHEN
- ◆ Checkpointing a job with the kill option (bchkpnt -k)
- ◆ Rerunnable job migration
- ◆ Job killed when an advance reservation has expired
- ◆ Remote lease job start fails

### Excluding LSF and user-related job exits

To explicitly *exclude* jobs exited because of user actions or LSF-related policies from the job exit calculation, set EXIT\_RATE\_TYPE = JOBEXIT\_NONLSF in `lsb.params`. JOBEXIT\_NONLSF tells LSF to include all job exits *except* those that are related to user action or LSF policy. This is the default value for EXIT\_RATE\_TYPE.

To *include* all job exit cases in the exit rate count, you must set EXIT\_RATE\_TYPE = JOBEXIT in `lsb.params`. JOBEXIT considers all job exits.

Jobs killed by signal external to LSF will still be counted towards exit rate

Jobs killed because of job control SUSPEND action and RESUME action are still counted towards the exit rate. This because LSF cannot distinguish between jobs killed from SUSPEND action and jobs killed by external signals.

If both JOBEXIT and JOBEXIT\_NONLSF are defined, JOBEXIT\_NONLSF is used.

### Local jobs

When EXIT\_RATE\_TYPE=JOBINIT, various job initialization failures are included in the exit rate calculation, including:

- ◆ Host-related failures; for example, incorrect user account, user permissions, incorrect directories for checkpointable jobs, host name resolution failed, or other execution environment problems
- ◆ Job-related failures; for example, pre-execution or setup problem, job file not created, etc.

### Parallel jobs

By default, or when EXIT\_RATE\_TYPE=JOBEXIT\_NONLSF, job initialization failure on the first execution host does not count in the job exit rate calculation. Job initialization failure for hosts other than the first execution host are counted in the exit rate calculation.

When EXIT\_RATE\_TYPE=JOBINIT, job initialization failure happens on the first execution host are counted in the job exit rate calculation. Job initialization failures for hosts other than the first execution host are *not* counted in the exit rate calculation.

---

**TIP:** For parallel job exit exceptions to be counted for *all* hosts, specify EXIT\_RATE\_TYPE=HPCINIT or EXIT\_RATE\_TYPE=JOBEXIT\_NONLSF JOBINIT.

---

### Remote jobs

By default, or when EXIT\_RATE\_TYPE=JOBEXIT\_NONLSF, job initialization failures are counted as exited jobs on the remote execution host and are included in the exit rate calculation for that host. To include only *local* job initialization failures on the execution cluster from the exit rate calculation, set EXIT\_RATE\_TYPE to include only JOBINIT or HPCINIT.

## Scaling and tuning job exit rate by number of slots

On large, multiprocessor hosts, use `ENABLE_EXIT_RATE_PER_SLOT=Y` in `lsb.params` to scale the job exit rate so that the host is only closed when the job exit rate is high enough in proportion to the number of processors on the host. This avoids having a relatively low exit rate close a host inappropriately.

Use a float value for `GLOBAL_EXIT_RATE` in `lsb.params` to tune the exit rate on multislot hosts. The actual calculated exit rate value is never less than 1.

### Example: exit rate of 5 on single processor and multiprocessor hosts

On a single-processor host, a job exit rate of 5 is much more severe than on a 20-processor host. If a stream of jobs to a single-processor host is consistently failing, it is reasonable to close the host or take some other action after 5 failures.

On the other hand, for the same stream of jobs on a 20-processor host, it is possible that 19 of the processors are busy doing other work that is running fine. To close this host after only 5 failures would be wrong because effectively less than 5% of the jobs on that host are actually failing.

### Example: float value for GLOBAL\_EXIT\_RATE on multislot hosts

Using a float value for `GLOBAL_EXIT_RATE` allows the exit rate to be less than the number of slots on the host. For example, on a host with 4 slots, `GLOBAL_EXIT_RATE=0.25` gives an exit rate of 1. The same value on an 8 slot machine would be 2 and so on. On a single-slot host, the value is never less than 1.

## For more information

- ◆ See [Handling Host-level Job Exceptions](#) on page 88 for information about configuring host-level job exceptions.
- ◆ See [Handling Job Exceptions in Queues](#) on page 100 for information about configuring job exceptions. in queues

## Handling Job Exceptions



# Managing Users and User Groups

## Contents

- ◆ [Viewing User and User Group Information](#) on page 129
- ◆ [About User Groups](#) on page 131
- ◆ [Existing User Groups as LSF User Groups](#) on page 131
- ◆ [LSF User Groups](#) on page 132

## Viewing User and User Group Information

You can display information about LSF users and user groups using the `busers` and `bugroup` commands.

The `busers` command displays information about users and user groups. The default is to display information about the user who invokes the command. The `busers` command displays:

- ◆ Maximum number of jobs a user or group may execute on a single processor
- ◆ Maximum number of job slots a user or group may use in the cluster
- ◆ Maximum number of pending jobs a user or group may have in the system.
- ◆ Total number of job slots required by all submitted jobs of the user
- ◆ Number of job slots in the `PEND`, `RUN`, `SSUSP`, and `USUSP` states

The `bugroup` command displays information about user groups and which users belong to each group.

The `busers` and `bugroup` commands have additional options. See the `busers(1)` and `bugroup(1)` man pages for more details.

---

**RESTRICTION:** The keyword `all` is reserved by LSF. Ensure that no actual users are assigned the user name "all."

---

## Viewing User and User Group Information

### View user information

- 1 Run `busers all`.

```
busers all
USER/GROUP JL/P MAX NJOBS PEND RUN SSUSP USUSP RSV
default 12 - - - - - - -
user9 1 12 34 22 10 2 0 0
groupA - 100 20 7 11 1 1 0
```

### View user pending job threshold information

- 1 Run `busers -w`, which displays the pending job threshold column at the end of the `busers all` output.

```
busers -w
USER/GROUP JL/P MAX NJOBS PEND RUN SSUSP USUSP RSV MPEND
default 12 - - - - - - - 10
user9 1 12 34 22 10 2 0 0 500
groupA - 100 20 7 11 1 1 0 200000
```

### View user group information

- 1 Run `bugroup`.

```
bugroup
GROUP_NAME USERS
testers user1 user2
engineers user3 user4 user10 user9
develop user4 user10 user11 user34 engineers/
system all users
```

### View user share information

- 1 Run `bugroup -l`, which displays user share group membership information in long format.

```
bugroup -l
GROUP_NAME: testers
USERS: user1 user2
SHARES: [user1, 4] [others, 10]

GROUP_NAME: engineers
USERS: user3 user4 user10 user9
SHARES: [others, 10] [user9, 4]

GROUP_NAME: system
USERS: all users
SHARES: [user9, 10] [others, 15]

GROUP_NAME: develop
```

```
USERS: user4 user10 user11 engineers/
SHARES: [engineers, 40] [user4, 15] [user10, 34] [user11,
16]
```

---

## About User Groups

User groups act as aliases for lists of users. The administrator can also limit the total number of running jobs belonging to a user or a group of users.

You can define user groups in LSF in several ways:

- ◆ Use existing user groups in the configuration files
- ◆ Create LSF-specific user groups
- ◆ Use an external executable to retrieve user group members

If desired, you can use all three methods, provided the user and group names are different.

## Existing User Groups as LSF User Groups

User groups already defined in your operating system often reflect existing organizational relationships among users. It is natural to control computer resource access using these existing groups.

You can specify existing UNIX user groups anywhere an LSF user group can be specified.

### How LSF recognizes UNIX user groups

Only group members listed in the `/etc/group` file or the file `group.byname` NIS map are accepted. The user's primary group as defined in the `/etc/passwd` file is ignored.

The first time you specify a UNIX user group, LSF automatically creates an LSF user group with that name, and the group membership is retrieved by `getgrnam(3)` on the master host at the time `mbatchd` starts. The membership of the group might be different from the one on another host. Once the LSF user group is created, the corresponding UNIX user group might change, but the membership of the LSF user group is not updated until you reconfigure LSF (`badmin`). To specify a UNIX user group that has the same name as a user, use a slash (`/`) immediately after the group name: `group_name/`.

### Requirements

UNIX group definitions referenced by LSF configuration files must be uniform across all hosts in the cluster. Unexpected results can occur if the UNIX group definitions are not homogeneous across machines.

### How LSF resolves users and user groups with the same name

If an individual user and a user group have the same name, LSF assumes that the name refers to the individual user. To specify the group name, append a slash (`/`) to the group name.

For example, if you have both a user and a group named `admin` on your system, LSF interprets `admin` as the name of the user, and `admin/` as the name of the group.

## Where to use existing user groups

Existing user groups can be used in defining the following parameters in LSF configuration files:

- ◆ `USERS` in `lsb.queues` for authorized queue users
- ◆ `USER_NAME` in `lsb.users` for user job slot limits
- ◆ `USER_SHARES` (optional) in `lsb.hosts` for host partitions or in `lsb.queues` or `lsb.users` for queue fairshare policies

## LSF User Groups

You can define an LSF user group within LSF or use an external executable to retrieve user group members.

Use `bugroup` to view user groups and members, use `busers` to view all users in the cluster.

## Where to use LSF user groups

LSF user groups can be used in defining the following parameters in LSF configuration files:

- ◆ `USERS` in `lsb.queues` for authorized queue users
- ◆ `USER_NAME` in `lsb.users` for user job slot limits
- ◆ `USER_SHARES` (optional) in `lsb.hosts` for host partitions or in `lsb.queues` for queue fairshare policies

If you are using existing OS-level user groups instead of LSF-specific user groups, you can also specify the names of these groups in the files mentioned above.

## Configure user groups

- 1 Log in as the LSF administrator to any host in the cluster.

- 2 Open `lsb.users`.

- 3 If the `UserGroup` section does not exist, add it:

```
Begin UserGroup
GROUP_NAME GROUP_MEMBER USER_SHARES
financial (user1 user2 user3) ([user1, 4] [others,
10])
system (all) ([user2, 10] [others,
15])
regular_users (user1 user2 user3 user4) -
part_time_users (!) -
End UserGroup
```

- 4 Specify the group name under the `GROUP_NAME` column.

External user groups must also be defined in the `egroup` executable.

- 5 Specify users in the `GROUP_MEMBER` column.

For external user groups, put an exclamation mark (!) in the `GROUP_MEMBER` column to tell LSF that the group members should be retrieved using `egroup`.

- 6 Optional: To enable hierarchical fairshare, specify share assignments in the `USER_SHARES` column.

- 7 Save your changes.
  - 8 Run `badadmin ckconfig` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
  - 9 Run `badadmin reconfig` to reconfigure the cluster.
- 

## External user group requirements (egroup)

An external user group is a user group for which membership is not statically configured, but is instead retrieved by running an external executable with the name `egroup`. The `egroup` executable must be in the directory specified by `LSF_SERVERDIR`.

This feature allows a site to maintain group definitions outside LSF and import them into LSF configuration at initialization time.

The `egroup` executable is an executable you create yourself that lists group names and users who belong to the group.

This executable must have the name `egroup`. When `mbatchd` is restarted, it invokes the `egroup` executable and retrieves groups and group members. The external executable `egroup` runs under the same account as `mbatchd`.

The `egroup` executable must write user names for the user groups to its standard output, each name separated by white space.

The `egroup` executable must recognize the following command, since `mbatchd` invokes external user groups with this command:

```
egroup -u user_group_name
```

where *user\_group\_name* is the name of the user group defined in the executable `egroup` along with its members, and the user group is specified in `lsb.users`.



## Platform LSF Licensing

### Contents

- ◆ [The LSF License File](#) on page 136
- ◆ [How LSF Permanent Licensing Works](#) on page 140
- ◆ [Installing a Demo License](#) on page 142
- ◆ [Installing a Permanent License](#) on page 144
- ◆ [Updating a License](#) on page 149
- ◆ [FLEXlm Basics](#) on page 151
- ◆ [Multiple FLEXlm License Server Hosts](#) on page 154
- ◆ [Partial Licensing](#) on page 156
- ◆ [Floating Client Licenses](#) on page 160
- ◆ [Troubleshooting License Issues](#) on page 166

## The LSF License File

You must have a valid license to run LSF. This section helps you to understand the types of LSF licenses and the contents of the LSF license file. It does not contain information required to install your license.

---

**TIP:** To learn about licensing license a cluster that includes Windows hosts, see *Using Platform LSF on Windows*.

---

### Evaluation (demo) license

You can use a demo license to install Platform LSF and get it running temporarily, then switch to the permanent license before the evaluation period expires with no interruption in service, as described in [Installing a Permanent License](#) on page 144.

Although there may be exceptions, a typical demo license:

- ◆ Is used during your free evaluation of LSF
- ◆ Expires on a preset calendar date (30 days after the license was generated)
- ◆ Is file-based (does not require Macrovision® FLEXlm™ software)
- ◆ Licenses all LSF products
- ◆ Allows an unlimited number of hosts to be LSF servers

### Permanent license

Although there may be exceptions, a typical permanent license:

- ◆ Is granted when you purchase LSF
- ◆ Licenses specific LSF products that you have purchased
- ◆ Limits the number of hosts allowed to be LSF servers
- ◆ Requires FLEXlm™ 7.2 or later
- ◆ Is keyed to one or more specific FLEXlm license server hosts
- ◆ Does not expire

### Enforcement of dual-core processor licenses on Linux

Dual-core processor hosts running Linux must be licensed by the `lsf_dualcore_x86` license feature.

Each dual core processor requires one standard LSF license and one `lsf_dualcore_x86` license.

Use `lshosts -l` to see the number of dual-core licenses enabled and needed. For example:

```
lshosts -l hostB
HOST_NAME: hostB
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1 2 1 2016M 1983M 72917M 0 Yes 1 2 2

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster LSF_Sched_Fairshare
LSF_Sched_Resource_Reservation LSF_Sched_Preemption LSF_Sched_Parallel
LSF_Sched_Advance_Reservation LSF_DualCore_x86)
```



```
LICENSE CLASS NEEDED: Class(B), Multi-cores
```

```
...
```

## Enforcement of multicore processor licenses on Linux and Windows

Multicore hosts running Linux or Windows must be licensed by the `lsf_dualcore_x86` license feature. Each physical processor requires one standard LSF license and `num_cores-1 lsf_dualcore_x86` licenses. For example, a processor with 4 cores requires 3 `lsf_dualcore_x86` licenses.

Use `lshosts -l` to see the number of multicore licenses enabled and needed. For example:

```
lshosts -l hostB
HOST_NAME: hostB
type model cpufl ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1 2 1 2016M 1983M 72917M 0 Yes 1 4 2

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster LSF_Sched_Fairshare
LSF_Sched_Resource_Reservation LSF_Sched_Preemption LSF_Sched_Parallel
LSF_Sched_Advance_Reservation LSF_DualCore_x86)
```

```
LICENSE CLASS NEEDED: Class(B), Multi-cores
```

```
...
```

## Enforcement of grid license management plugin licenses

The new license optimization features enabled by the grid license management plugin require the `lsf_mv_grid_filter` license feature.

The number of `lsf_mv_grid_filter` licenses should be at least the number of LSF License Scheduler licenses.

## Banded licensing

You can use permanent licenses with restrictions in operating system and hardware configurations. These banded licenses have three classes, with the E-class licenses having no restrictions. Banded licenses support the following operating systems and hardware configurations:

| License type | Supported operating systems               | Processor                      | Physical memory                                     | Physical processors/sockets      |
|--------------|-------------------------------------------|--------------------------------|-----------------------------------------------------|----------------------------------|
| B-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | Up to and including 4 GB physical memory on a node  | Up to and including 2 processors |
| S-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | Up to and including 16 GB physical memory on a node | Up to and including 4 processors |
| E-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | More than 16 GB physical memory on a node           | More than 4 processors           |
|              | All other LSF-supported operating systems | Intel X86/AMD64/EM64T          | N/A                                                 | N/A                              |
|              | N/A                                       | All other supported processors | N/A                                                 | N/A                              |

## The LSF License File

### In the LSF license file:

```
FEATURE lsf_manager lsf_ld 6.200 8-may-2008 2 ADE2C12C1A81E5E8F29C \
VENDOR_STRING=Platform NOTICE=Class(S)
FEATURE lsf_manager lsf_ld 6.200 8-may-2008 10 1DC2C1CCEF193E42B6DC \
VENDOR_STRING=Platform NOTICE=Class(E)
```

### Determining what licenses a host needs

Use `lim -t` and `lshosts -l` to see the license requirements for a host. For example:

```
lim -t hostA
Host Type : NTX64
Host Architecture : EM64T_1596
Physical Processors : 2
Cores per Processor : 4
Threads per Core : 2
License Needed : Class(B), Multi-core
Matched Type : NTX64
Matched Architecture : EM64T_3000
Matched Model : Intel_EM64T
CPU Factor : 60.0

lshosts -l hostA
HOST_NAME: hostA
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1 2 1 2016M 1983M 72917M 0 Yes 2 4 2
...
 LICENSE CLASS NEEDED: Class(B), Multi-cores
...
```

### Format of the demo license file

This is intended to familiarize you with the demo license file. You do not need to read this section if you are only interested in installing the license.

LSF licenses are stored in a text file. The default name of the license file is `license.dat`.

The `license.dat` file for an LSF license normally contains the same products defined in `lsf.cluster.cluster_name`.

The `license.dat` file for a demo license contains a `FEATURE` line for each LSF product. Each feature contains an expiry date and ends with the string `DEMO`.

For example:

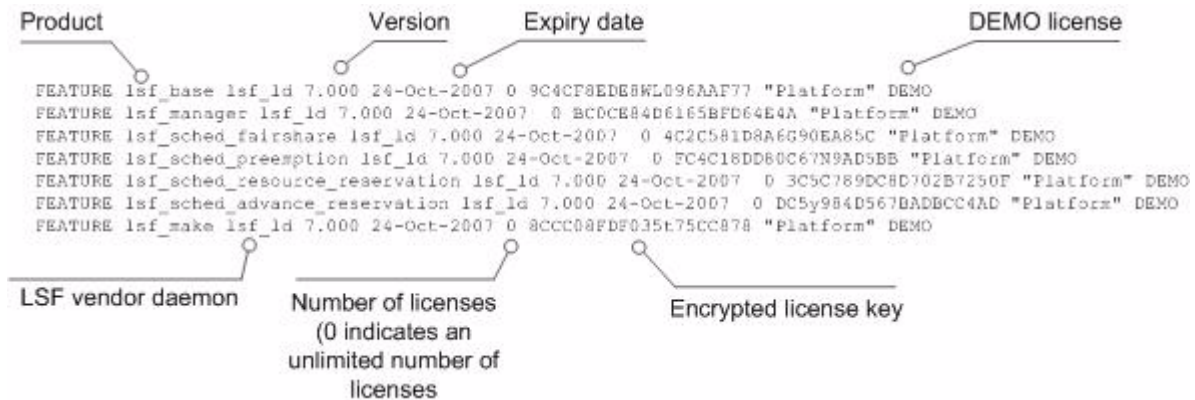
```
FEATURE lsf_base lsf_ld 7.000 24-Oct-2008 100 DCF7C3D92A5471A12345 "Platform" DEMO
```

The `FEATURE` line contains an encrypted key to prevent tampering.

A demo license does not require a server daemon or vendor daemon, so it does not contain `SERVER` or `DAEMON` lines, only `FEATURE` lines.

## Example demo license file

The following is an example of a demo license file. This file licenses LSF 7, advance reservation, and Platform Make. The license is valid until October 24, 2008.



## Format of the permanent license file

A permanent license file has the same format as other products licensed with FLEXlm. If you are already familiar with FLEXlm license files, you can skip this section.

In addition to the information presented in the demo license file (see [Format of the demo license file](#) on page 138), the permanent license file includes the following:

- ◆ A **SERVER** line for each FLEXlm server host. The **SERVER** line contains the following server information:
  - ❖ Host name
  - ❖ Hardware host ID
  - ❖ TCP port number used by the FLEXlm license server daemon (`lmgrd`)

```
SERVER hostA 880a0748a 1700
```
- ◆ A **DAEMON** line for each software vendor, which gives the file path name of the LSF license vendor daemon (`lsf_ld`, normally installed in `LSF_SERVERDIR`).
 

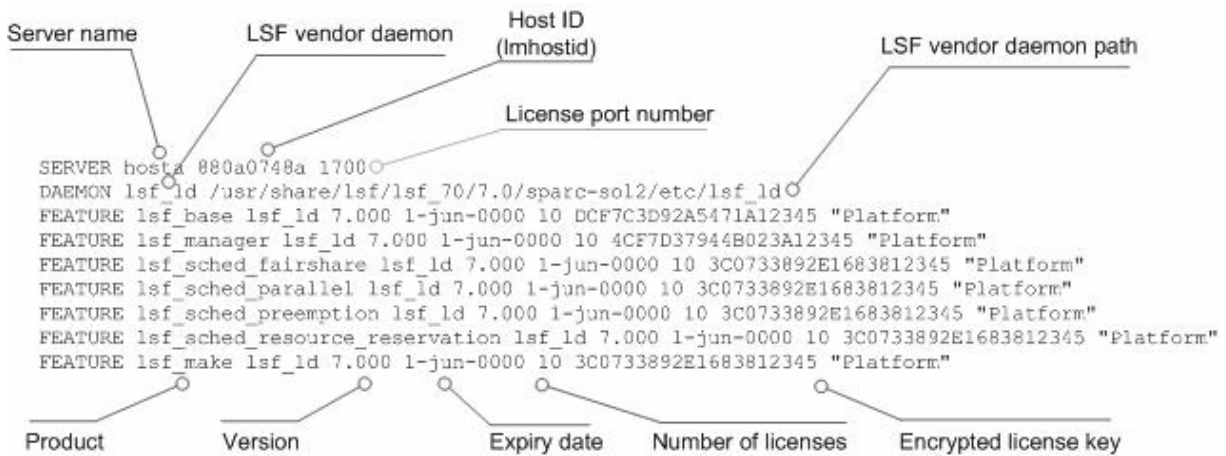
```
DAEMON lsf_ld /usr/local/lsf/etc/lsf_ld
```
- ◆ Each **FEATURE** line in the license ends with **Platform**, instead of **DEMO**.

```
FEATURE lsf_base lsf_ld 7.000 1-jan-0000 100 DCF7C3D92A5471A12345 "Platform"
```

For permanent licenses, the licenses granted by the **FEATURE** line can be accessed only through license server hosts listed on the **SERVER** lines.

## Example permanent license file

The following is an example of a permanent license file.



The license server daemon is configured to run on `hosta`, using TCP port 1700. It allows 10 single-processor hosts to run Platform LSF 7 and Platform Make, with no expiry date.

## How LSF Permanent Licensing Works

This section is intended to give you a better understanding of how LSF licensing works in a production environment with a permanent license. It does not contain information required to install your license.

Platform LSF uses the `FLEXlm` license management product from GLOBETrotter Software to control its licenses. LSF licenses are controlled centrally through the LSF master LIM.

### FLEXlm license server

Permanent LSF licenses are managed by the `FLEXlm` license server daemon (`lmgrd`). The `FLEXlm` license server daemon runs on a license server host you choose (for failover purposes, the daemon can run on multiple hosts).

The `lmgrd` daemon starts the LSF vendor license daemon `lsf_ld`, which periodically keeps track of how many LSF licenses are checked out and who has them. Only one `lsf_ld` can run on a host. If `lsf_ld` stops running, `lmgrd` immediately stops serving LSF licenses to all LSF hosts.

The LIM on the LSF master hosts contacts the license server host to get the necessary LSF licenses. It then propagates licenses to all LSF server hosts and client hosts. Multiple LSF clusters can get licenses from the same license server host.

The `TIMEOUT ALL` parameter in the `FLEXlm` license option file changes timeout values, including how quickly the master host releases licenses during failover. LSF supports a minimum timeout value of 15 minutes. For information about how to configure the `TIMEOUT ALL` parameter, see the `FLEXlm` documentation.

## LSF license checkout

Only the master LIM can check out licenses. No other part of LSF has any contact with the *FLEXlm* license server daemon. Once LIM on the master host identifies itself as the master, it reads the `LSF_CONFDIR/lsf.cluster.cluster_name` file to get the host information to calculate the total number of licenses needed. Most LSF software is licensed per CPU, not per host or per cluster, so multi-processor hosts require multiple LSF licenses.

After the cluster is properly licensed, the master LIM contacts the license server daemon periodically to confirm the availability of checked out LSF licenses.

LIM distributes the licenses needed this way:

- 1 Calculate the total number of licenses needed for the master LIM.
- 2 Before slave LIMs contact the master, calculate the total number of licenses needed for all LSF server hosts and check them out. When the slave LIMs start, they contact the master host to get the licenses they need.
- 3 Check out licenses needed for client hosts listed in `LSF_CONFDIR/lsf.cluster.cluster_name`. If the license checkout fails for any host, that host is unlicensed. The master LIM tries to check out the license later.

## LSF license grace period

If the master LIM finds the license server daemon has gone down or is unreachable, LSF has a grace period before the whole cluster is unlicensed. As long as the master LIM that originally received the licenses is not restarted or shut down, the LSF cluster can run up to 60 hours without licenses. If you reconfigure LSF after the license server daemon becomes unavailable, you lose the grace period and the cluster is unlicensed because the original LIM that carries the correct license information is killed and restarted during reconfiguration. This prevents LSF from becoming a single point of failure and enables LSF to function reliably over an extended period of time (for example, over a long weekend) should the license server daemon fail.

## Unlicensed cluster

While LSF cannot contact a license server daemon, LSF commands are automatically resubmitted, not aborted.

## Installing a Demo License

This section includes instructions for licensing LSF with a new demo license.

Most new users should follow the procedure under [Install and license LSF for the first time](#) on page 142.

If you already have LSF installed, see [Install a demo license manually](#) on page 142.

### Install and license LSF for the first time

If LSF has never been installed before, you should install and license LSF in one step, using a demo license and the LSF installation program for UNIX, `lsfinstall`.

- 
- 1 Acquire your demo license before you install LSF.  
See [Get a demo license](#) on page 143.
  - 2 When you receive your license file, save it as `license.dat`.  
See [Viewing and editing the license file](#) on page 145.
  - 3 Install LSF using `lsfinstall` as described in *Installing Platform LSF on UNIX and Linux*. `lsfinstall` automatically sets up the LSF demo license.
- 

### Install a demo license manually

If you just need to update or replace an existing LSF license, see [Updating a License](#) on page 149.

If LSF is installed without a license file, or the license file is not properly installed, you can install a demo license manually.

- 
- 1 Acquire your demo license.  
See [Get a demo license](#) on page 143.
  - 2 When you receive your license file, save it as `license.dat`.  
See [Viewing and editing the license file](#) on page 145.
  - 3 Move the license file to a location where it can be shared.  
See [Location of the LSF license file for a demo license](#) on page 143.
  - 4 Set the `LSF_LICENSE_FILE` parameter to point to your license file.  
See [LSF\\_LICENSE\\_FILE parameter](#) on page 146.
  - 5 Start or reconfigure LSF. This causes the license file to be read and the changes accepted by LSF:
    - ❖ If LSF daemons are already running, reconfigure LSF using the following LSF commands:

```
lsadmin reconfig
badmin reconfig
```
    - ❖ If this is a new installation, start LSF using one of the following two methods:
      - a On the LSF master host, run the following LSF commands:

```
lsadmin limstartup all
```

```
lsadmin resstartup all
badmin hstartup all
```

- b** On any LSF host, run the script:  
LSF\_BINDIR/lssfstartup
- 

## Get a demo license

To get a demo license from Platform Computing or your Platform LSF vendor.

## Location of the LSF license file for a demo license

For a demo license, each LSF host must be able to read the license file.

The installation program `lsfinstall` puts the LSF license file in a shared directory where it is available to all LSF hosts.

If you install the license manually, use either of the following methods to ensure that a license is available to all hosts:

- ◆ Share the same license file between all the hosts using NFS
- ◆ Install a separate copy of the license file on each host

## Installing a Permanent License

This section includes instructions for licensing LSF with a new permanent license. If you have not yet installed LSF, you can use a demo license to get started. See [Installing a Demo License](#) on page 142.

If you already have LSF, see [Install a permanent license for the first time](#) on page 144.

### Install a permanent license for the first time

If you are switching from a demo license to a permanent license, follow these instructions to set up the permanent license. You can discard the old demo license; LSF cannot use both licenses at the same time.

If you just need to update an existing permanent license, see [Updating a License](#) on page 149.

- 1 Acquire your permanent license.

See [Getting a permanent license](#) on page 145.

- 2 When you receive your license file, save it as `license.dat`.

See [Viewing and editing the license file](#) on page 145.

- 3 Edit the DAEMON line in the license file to point to the LSF vendor license daemon `lsf_ld`.

The LSF vendor license daemon is installed in `LSF_SERVERDIR` (defined in `lsf.conf` or set in your environment). For example:

```
DAEMON lsf_ld /usr/share/lsf/lsf_62/7.0/sparc-sol2/etc/lsf_ld
```

The `lsf_ld` binary should be available to the `FLEXlm` server using this path.

- 4 Verify that the LSF products enabled by the PRODUCTS line in `LSF_CONFDIR/lsf.cluster.cluster_name` are licensed by features in the license file.

For example, if the PRODUCTS line contains:

```
PRODUCTS=LSF_Make LSF_MultiCluster
```

then your license must include FEATURE lines such as:

```
FEATURE lsf_make lsf_ld 7.000 1-jun-0000 10 DCF7C3D92A5471A12345 "Platform"
FEATURE lsf_multicluster lsf_ld 7.000 1-jun-0000 10 4CF7D37944B023A12345 "Platform"
```

If you do not have licenses for some products in the PRODUCTS line, contact Platform Computing or your Platform LSF vendor. To continue installing your permanent license, remove the unlicensed products from the PRODUCTS line.

See [Licensing LSF products and features](#) on page 147.

- 5 Make sure the file is in a location where it can be accessed by the license server daemons.

See [Location of the LSF license file for a permanent license](#) on page 146.

- 6 Set the `LSF_LICENSE_FILE` parameter to point to your license file.

See [LSF\\_LICENSE\\_FILE parameter](#) on page 146.

- 7 Start the license server daemon.



See [Start the license daemons](#) on page 151.

- 8 To allow the new permanent license to take effect, reconfigure the cluster:
    - a `lsadmin reconfig`
    - b `badmin reconfig`
  - 9 After the cluster starts, use the following commands to make sure LSF is up and running:
    - a `lsid`
    - b `bhosts`
- 

## Getting a permanent license

To install Platform LSF for production use, you must get a permanent license from Platform or your LSF vendor.

Platform creates a permanent license that is keyed to the license server host or hosts. Some host types have a built-in hardware host ID; on others, the hardware address of the primary LAN interface is used. For a permanent license to be created, you must supply a server host name and the hardware host identifier for each license server host at your site.

Send the following information to Platform Computing or your Platform LSF vendor.

- ◆ Host name of the license server host (see [FLEXlm license server host](#) on page 147)
- ◆ Host identifier of the license server host (see [Getting the FLEXlm license server host identifier](#) on page 145)
- ◆ Products required (see [Licensing LSF products and features](#) on page 147)
- ◆ Number of licenses required for your cluster (see [LSF license checkout](#) on page 141)

## Getting the FLEXlm license server host identifier

When an LSF license is managed by FLEXlm, you must provide a hardware host name and host identifier for the FLEXlm license server host at your site.

If you do not already use FLEXlm to manage other applications, you must choose a host as the FLEXlm license server host before you request your license. See [Selecting a license server host](#) on page 148.

Use the `lmhostid` command (normally located in `LSF_SERVERDIR`) to get the hardware identifier of your FLEXlm license server host. For example, run this command on the FLEXlm server host:

```
lmhostid
lmhostid - Copyright (C) 1989-1997 Globetrotter Software, Inc.
The FLEXlm host ID of this machine is "68044d20"
```

In this example, send the code “68044d20” to Platform.

## Viewing and editing the license file

Your LSF license should be a text file (normally named `license.dat`). Use any text editor such as `vi` or `emacs` to open a copy of your license file for viewing or editing.

## Installing a Permanent License

For example,

- ◆ If you receive your license from Platform as text, you must create a new file and copy the text into the file.
- ◆ You might have to modify lines in the license, such as the path in the DAEMON line when you install a new permanent license.
- ◆ You might want to check that the license includes the correct features before you install it.
- ◆ You might want to merge the contents of the LSF license into a single file that includes licenses for all applications that use FLEXlm.

If you can, make carriage returns visible, or view the text without word wrap. Although the lines might wrap when displayed on a screen, make sure each line in the text file has no extra characters. You can accidentally corrupt your license file if you view it or copy it from email, and then save it with hidden line breaks.

Do not try to modify lines or fields unless instructed to do so by Platform. You could corrupt the license. Do not combine demo license lines with permanent license lines. For more information about LSF license files, see [Format of the demo license file](#) on page 138 and [Format of the permanent license file](#) on page 139.

## Location of the LSF license file for a permanent license

For a permanent license, the FLEXlm license daemon `lmgrd` and the LSF vendor daemon `lsf_ld` must be able to read the LSF license file. You can put the license file on the license server host, or in a shared directory.

Daemons on the LSF master host do not need any access to the permanent license file.

## LSF\_LICENSE\_FILE parameter

The `LSF_LICENSE_FILE` parameter in `LSF_CONFDIR/lsf.conf` points to the LSF license file.

The installation program `lsfinstall` configures the `LSF_LICENSE_FILE` parameter automatically for demo licenses only. You must set `LSF_LICENSE_FILE` manually if you do either of the following:

- ◆ Install a permanent license
- ◆ Install a DEMO or permanent license manually and change the location of the license file

To configure `LSF_LICENSE_FILE`, specify the full path name to the license file. A permanent license file should also be visible to the FLEXlm license server host using the same path.

For example, after you run `lsfinstall`, the default setting is:

`LSF_LICENSE_FILE` can also be the name of the license server host and the port number used by `lmgrd` in the form `port_number@host_name`. For example, if your license file contains the line:

```
SERVER hosta 68044d20 1700
```

`LSF_LICENSE_FILE` would be:

```
LSF_LICENSE_FILE="1700@hosta"
```

**FLEXlm default** If you installed FLEXlm separately from LSF to manage other software licenses, the default FLEXlm installation puts the license file in the following location:  
`/usr/local/flexlm/licenses/license.dat`

**Troubleshooting** If this parameter points to an older or incorrect license key, correct the problem using one of these two methods:

- ◆ Change the path to point to the location of the new key.
- ◆ Put the new key in the location specified by the path (make a backup copy of your old license key before you overwrite it).

## Licensing LSF products and features

All LSF software requires a license. Some LSF features are enabled by the license file alone, but other products must also be included in the cluster configuration file, or the FEATURE line in the license file is ignored. However, if you already have the FEATURE line in your license file, you can install or enable the corresponding products later on.

The following strings are examples of what can be listed in the PRODUCTS line in the Parameters section of the `lsf.cluster.cluster_name` file, to indicate which LSF products that the cluster should run.

- ◆ LSF\_Base
- ◆ LSF\_Manager
- ◆ LSF\_Sched\_Fairshare
- ◆ LSF\_Sched\_Preemption
- ◆ LSF\_Sched\_Parallel
- ◆ LSF\_Sched\_Resource\_Reservation
- ◆ LSF\_Make
- ◆ LSF\_MultiCluster
- ◆ LSF\_Sched\_Advance\_Reservation

If these products are listed in the cluster configuration, the LSF license must also include FEATURE lines for these products.

In addition, there are some “extra” licensed features that do not have a matching item in the PRODUCTS line. Do not remove features from your license unless instructed to do so by Platform. For example, the following strings are valid in the license file, but should not be used in the PRODUCTS line:

- ◆ LSF\_Client
- ◆ LSF\_Float\_Client

LSF client hosts are licensed per host, not per CPU, so there is no difference between licensing a single-processor host and a multi-processor host.

See [Floating Client Licenses](#) on page 160 for information about configuring LSF floating clients.

## FLEXlm license server host

A permanent LSF license is tied to the host ID of a particular license server host and cannot be used on another host.

## Installing a Permanent License

If you are already running *FLEXlm* to support other software licenses, you can use the existing license server host to manage LSF also. In this case, you will add your Platform LSF license key to the existing *FLEXlm* license file.

If you are not already using *FLEXlm*, or prefer to administer LSF license management separately, you must choose a host to run the license daemons. See [Selecting a license server host](#) on page 148.

It is possible to run multiple license server hosts for failover purposes. See [Multiple FLEXlm License Server Hosts](#) on page 154.

## Selecting a license server host

By reading this, you will gain the information needed to make an educated decision when selecting a license server host.

The *FLEXlm* license server daemon normally runs on one host. LSF tolerates failure of the license server daemon for up to 60 hours, as long as the master LIM is not restarted or shut down.

If you are installing a permanent license, choose a reliable host as the license server host to ensure that the LSF licenses are always available. LSF cannot run if it cannot contact the license server daemon. Although the license server host can be an LSF host, it is usually a host outside of the cluster. The license daemons create very little load, so they can be run on the host that is the dedicated file server for the Platform LSF software. This permits the licenses to be available whenever the LSF software is available.

You should not make the license server host the same as the master host for the cluster. If you do this, and the master host goes down, the backup master that takes over will not be able to check license tokens out from the license server daemon on the original master which has failed.

## FLEXlm software for the license server host

Permanent (server-based) LSF licenses work with *FLEXlm* version 7.2 or later.

If your *FLEXlm* license server host is of the same host type as one or more LSF hosts, the *FLEXlm* software is included in the LSF distribution and automatically installed under `LSF_SERVERDIR`, which is a shared directory (so there is no requirement to copy any software to your *FLEXlm* license server host; just include `LSF_SERVERDIR` in your `PATH` environment variable on the license server host so that you can access the files and start the daemons).

If your *FLEXlm* license server host is a different host type, you do not need the complete LSF distribution. You can download just the *FLEXlm* software from Platform's FTP site, and copy it to any convenient location.

## Updating a License

This section is intended for those who are updating an existing LSF license file.

To switch your demo license to a permanent license, see [Installing a Permanent License](#) on page 144.

To update a license:

- 1 Contact Platform to get the license. See [Requesting a new license](#) on page 149.
- 2 Update the license using one of the following procedures
  - ❖ [Updating a license with FEATURE lines](#) on page 149
  - ❖ [Update a license with INCREMENT lines](#) on page 150

---

**REMEMBER:** After updating an existing LSF license file or adding FLEXlm licenses, you must use `lsadmin limrestart` on the master LIM or `lsadmin reconfig` from any LIM to use the new licenses.

---

## Requesting a new license

To update your license, contact Platform Computing or your Platform LSF vendor. Since you already have a license, you will only receive new lines to put into your existing file.

### FEATURE LINES

To update your license file, LSF licenses are sent to you in the form of FEATURE license lines when you:

- ◆ Already have some LSF products, but purchase a new product for the first time
- ◆ Upgrade LSF to a newer version
- ◆ Already have LSF, but time-limited licenses have expired

### INCREMENT LINES

If you already have an LSF product, licenses for additional hosts are normally sent to you in the form of INCREMENT license lines.

## Updating a license with FEATURE lines

FLEXlm only accepts one license key for each feature listed in a license file. If there is more than one FEATURE line for the same feature, only the first FEATURE line is used.

If you received one or more FEATURE lines, update your license by adding the lines to your existing license file.

- 1 Edit your `license.dat` file using a text editor like `vi` or `emacs`.  
See [Viewing and editing the license file](#) on page 145.
- 2 You should always have just one FEATURE line for each LSF product:
  - ❖ If this is the first time you have installed the product, append the FEATURE line to your existing license file (if you wish, you can insert it anywhere after the SERVER line).
  - ❖ If you already have a license for the product, replace the old FEATURE line with the new line.

## Updating a License

- 3 If you want LSF 4.x and LSF 5.x clusters to share a license file, make sure your license includes the `FEATURE` line for `lsf_batch` version 4.x.
- 4 Reconfigure LSF using either of the following LSF commands:
  - ❖ `lsadmin reconfig`
  - ❖ `lsadmin restart` on the master LIM

The license file is re-read and the changes accepted by LSF. At this point, the LSF license has been updated. However, some products may also require installation or upgrade of LSF software before you can use the new functionality.

---

## Update a license with INCREMENT lines

- 1 If you received one or more `INCREMENT` lines, update your license by adding the lines to your existing license file.
- 2 Edit your `license.dat` file using a text editor like `vi` or `emacs`.  
See [Viewing and editing the license file](#) on page 145.
- 3 Always append an `INCREMENT` line, do not overwrite or delete existing license lines in the process.
  - ❖ If this is the first increment, add the `INCREMENT` line for each product after the `FEATURE` line for that product.
  - ❖ If you already have an `INCREMENT` line for the product, add the second `INCREMENT` line after the first, and so on.
- 4 Reconfigure LSF using either of the following LSF commands:
  - ❖ `lsadmin reconfig`
  - ❖ `lsadmin restart` on the master LIM

The license file is re-read and the changes accepted by LSF.

---

## FLEXlm Basics

This section is for users installing a permanent license, as FLEXlm is not used with demo licenses. Users who already know how to use FLEXlm will not need to read this section.

FLEXlm is used by many UNIX software packages because it provides a simple and flexible method for controlling access to licensed software. A single FLEXlm license server daemon can handle licenses for many software packages, even if those packages come from different vendors. This reduces the system's administration load, since you do not need to install a new license manager every time you get a new package.

### Start the license daemons

FLEXlm uses license daemons to manage permanent licenses. For a brief description of FLEXlm and its license daemons, see [FLEXlm license server](#) on page 140.

This is a procedure that describes how to start the FLEXlm license daemons.

- 1 Log on to the license server host as LSF administrator.

---

**IMPORTANT:** Do not run `lmgrd` as root.

---

- 2 If you have an old `lsf_lsd` running, run `lmdown` to kill it.  
You can only have one `lsf_lsd` daemon running on a host.
- 3 Run the `lmgrd` command in `LSF_SERVERDIR` to start the license server daemon:

```
lmgrd -c /usr/share/lsf/lsf_62/conf/license.dat -l
/usr/share/lsf/lsf_62/log/license.log
```

The `-c` option specifies the license file (or license file list, if you have multiple license server hosts). For more information, see [LSF\\_LICENSE\\_FILE parameter](#) on page 146.

The `-l` option specifies the debug log path. For more information, see [FLEXlm log file](#) on page 152.

---

**TIP:** You should include `LSF_SERVERDIR` in your `PATH` environment variable. You may want to include the full command line in your system startup files on the license server host, so that `lmgrd` starts automatically during system reboot.

---

See [Checking the license server status](#) on page 151 to check the status of `lmgrd`.

---

### Checking the license server status

If you are using a permanent LSF license, use the `lmstat` command to check the status of the license server daemon. This check can tell you whether or not your attempt to start your license server daemon succeeded. If your attempt failed, see [lmgrd fails with message "Port already in use"](#) on page 168.

The `lmstat` command is in `LSF_SERVERDIR`. For example:

```
/usr/share/lsf/lsf_62/7.0/sparc-sol2/etc/lmstat
```

Run `lmstat -a -c LSF_LICENSE_FILE` from the FLEXlm license server and also from the LSF master host. You must use the `-c` option of `lmstat` to specify the path to the LSF license file.

The output of `lmstat` gives the status of:

- ◆ The license server daemon (`lmgrd`)
- ◆ The LSF vendor daemon (`lsf_ld`)
- ◆ The number of available licenses for each product in the license file

For example, depending on the LSF features installed, the output of the command should look something like the following:

```
lmstat -a -c $LSF_ENVDIR/license.dat
lmstat - Copyright (C) 1989-1997 Globetrotter Software, Inc.
Flexible License Manager status on Fri 10/15/1999 13:23

License server status: 1711@hostA
 License file(s) on hostA: /usr/local/cluster1/mnt/conf/license.dat:

 hostA: license server UP (MASTER) v5.12

Vendor daemon status (on hostA):

 lsf_ld: UP v5.12

Feature usage info:
Users of lsf_base: (Total of 50 licenses available)

 "lsf_base" v4.100, vendor: lsf_ld
 floating license

 root hostB /dev/tty (v3.0) (hostA/1711 401), start Thu 10/14 12:32, 20 licenses
 ...
```

## FLEXlm log file

Read this to familiarize yourself with the FLEXlm log file.

The FLEXlm license server daemons log messages about the state of the license server hosts, and when licenses are checked in or out. This log helps to resolve problems with the license server hosts and to track license use. The log file grows over time. You can remove or rename the existing FLEXlm log file at any time.

You must choose a location for the log file when you start the license daemon. If you already have FLEXlm server running for other products and Platform LSF licenses are added to the existing license file, then the log messages for FLEXlm should go to the same log file you set up for other products. If FLEXlm is dedicated to managing LSF licenses, you can put the FLEXlm log in the same directory as your other system logs, or in the `/tmp` directory.



## License management utilities

FLEXlm provides several utility programs for managing software licenses. These utilities and their man pages are included in the Platform LSF software distribution.

Because these utilities can be used to shut down the FLEXlm license server daemon, and can prevent licensed software from running, they are installed in the `LSF_SERVERDIR` directory. For security reasons, this directory should only be accessible to LSF administrators. Set the file permissions so that only root and members of group 0 can use them.

LSF installs the following FLEXlm utilities in `LSF_SERVERDIR`:

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <b>lmcksum</b>  | Calculate check sums of the license key information                       |
| <b>lmdown</b>   | Shut down the FLEXlm server                                               |
| <b>lmhostid</b> | Display the hardware host ID                                              |
| <b>lmremove</b> | Remove a feature from the list of checked out features                    |
| <b>lmreread</b> | Tell the license daemons to re-read the license file                      |
| <b>lmstat</b>   | Display the status of the license server daemons and checked out licenses |
| <b>lmver</b>    | Display the FLEXlm version information for a program or library           |

For complete details on these commands, see the FLEXlm man pages.

## Multiple FLEXlm License Server Hosts

This section applies to permanent licenses only. Read this section if you are interested in the various ways you can distribute your licenses. This is valuable if you are interested in having some form of backup in case of failure. Compare with [Selecting a license server host](#) on page 148 to make an educated decision.

Although it is not necessary, you may want to understand how the FLEXlm license server behaves prior to setting up your license server hosts. For a brief description on how FLEXlm works, see [FLEXlm license server](#) on page 140.

If you are concerned about the reliability of your license server host, you can distribute the LSF licenses across multiple FLEXlm license server hosts. If one license server host goes down, LSF will not lose all of the available licenses. There are two ways to configure multiple license server hosts:

- ◆ Multiple license files with multiple license server hosts. For more information, see [Distributed license server hosts](#) on page 154.
- ◆ Single license file with three redundant license server hosts. For more information, see [Redundant license server hosts](#) on page 155.

### Distributed license server hosts

Configuring multiple license server hosts is optional. It provides a way to keep LSF running if a license server host goes down. There are two ways to configure multiple license servers. This section describes distributed license server hosts. See [Redundant license server hosts](#) on page 155 for information on the other configuration.

Distributing licenses over multiple server hosts provides a fallback, in case your license server daemons fail.

With this method, you run multiple license server daemons, each with its own license file. Each license file has a `SERVER` line keyed to the license server host it is assigned to. The cluster is partially licensed as long as any one license server daemon is running, and fully licensed when all license server daemons are running. When a license server host is unavailable, the licenses managed by that host are unavailable. You decide how many LSF licenses to put on each license server host.

### Enable multiple license server hosts

See the procedures for installing and configuring a permanent license. There are a few differences when you use distributed license server hosts:

- 1 See [Getting a permanent license](#) on page 145. You must obtain multiple license files, with your total number of licenses divided appropriately among the license server hosts. You must provide the following information for each license server host:
  - ❖ Host name and FLEXlm host ID
  - ❖ The products and number of licenses you want to be managed by this host
- 2 See [LSF\\_LICENSE\\_FILE parameter](#) on page 146. Specify the location of all the licenses in `LSF_LICENSE_FILE`, not just one. Use a pipe (`|`) to separate each location. List the primary license server host first (the one you want LSF to contact first).

- 3 See [Start the license daemons](#) on page 151. Start `lmgrd` on all license server hosts, not just one.
- 4 To allow the new permanent licenses to take effect, reconfigure the cluster with the commands:
  - ❖ `lsadmin reconfig`
  - ❖ `badmin reconfig`

## Redundant license server hosts

Configuring multiple license server hosts is optional. It provides a way to keep LSF running if a license server host goes down. There are two ways to configure multiple license servers. This section describes redundant license server hosts. See [Distributed license server hosts](#) on page 154 for information on the second configuration.

A permanent license key is tied to a particular license server host with a specific host ID. If that host is down, the license service is not available and LSF becomes unlicensed if the master LIM is shut down or restarted.

To prevent down time, you can configure three hosts as license server hosts. The license server daemon (`lmgrd`) and LSF vendor license daemon (`lsf_ld`) run on each license server host. With three redundant server hosts, if any one host is down, the other two continue to serve licenses. If any two hosts are down, the license service stops.

### Enable multiple license server hosts

See the procedures for installing and configuring a permanent license. There are a few differences when you use redundant license server hosts:

- 1 See [Getting a permanent license](#) on page 145. You must obtain a license file that contains three `SERVER` lines. You must provide the following information for each license server host:
  - ❖ Host name and `FLEXlm` host ID
- 2 See [LSF\\_LICENSE\\_FILE parameter](#) on page 146. Specify the location of all the licenses in `LSF_LICENSE_FILE`, not just one. Use a pipe (`|`) to separate each location. List the primary license server host first (the one you want LSF to contact first).
- 3 See [Start the license daemons](#) on page 151. Start `lmgrd` on all license server hosts, not just one.
- 4 To allow the new permanent licenses to take effect, reconfigure the cluster with the commands:
 

```
lsadmin reconfig
badmin reconfig
```

## Partial Licensing

This section applies to permanent licenses. Read this if you have a cluster in which not all of the hosts will require licenses for the same LSF products. In this section, you will learn how to save money through distributing your licenses efficiently.

Not all hosts in the cluster need to be licensed for the same set of LSF products. For example, some hosts might be licensed only for Platform Make, while others may also be licensed to run Platform MultiCluster. All hosts in the cluster remain licensed regardless of the license configuration of the rest of the cluster. If hosts become unavailable or new hosts are added, licenses are redistributed according to the new configuration.

This allows you to purchase only as many licenses as you need, rather than enabling the entire cluster for products that are only needed by a few hosts.

However, many LSF products do not support partial licensing. They must be enabled for the entire cluster, or not at all.

### Setting priority for license distribution

This describes how to define the order your licenses are given out to hosts.

To enable LSF server hosts to run partially licensed LSF products, edit the `Host` section of `LSF_CONFDIR/lsf.cluster.cluster_name` and include the product names in the `RESOURCES` column for specific hosts. When the LSF cluster starts, the master LIM reads the `lsf.cluster.cluster_name` file and determines the LSF products that each host is licensed to use.

For a permanent license, the license manager retrieves the appropriate licenses for the cluster, and distributes the licenses to the hosts in the order they are listed in `lsf.cluster.cluster_name`. You can see the order in which licenses are distributed with the command `lshosts`.

### Displaying licensed products

This describes how to view what products are licensed for any host in the cluster.

Use the `lshosts -l` command.

```
lshosts -l hostA
HOST_NAME: hostA
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2

RESOURCES: Not defined
RUN_WINDOWS: (always open)

Licenses enabled: (LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation)

LOAD_THRESHOLDS:
 r15s r1m r15m ut pg io ls it tmp swp mem tmp2 nio console
 - 3.5 - - - - - - - - - - - 0.0
```

## Example of partial licensing

Here is an example that will allow you to better visualize the concept of partial licensing. Through this example, you can learn how to configure your hosts to use partial licensing.

### Scenario

In the following configuration, the license file contains licenses for LSF, Platform Make and Platform LSF MultiCluster. The licenses have the following distribution:

- ◆ 3 LSF
- ◆ 1 Platform Make
- ◆ 1 Platform MultiCluster

All three single-CPU hosts in the cluster are licensed for LSF, while `hostA` and `hostC` are licensed for Platform Make. The `hostB` is explicitly licensed for Platform MultiCluster, and not for Platform Make. The `RESOURCES` field in the `Host` section of `lsf.cluster.cluster_name` must contain the LSF products `LSF_Base` and `LSF_Manager`, in addition to `LSF_MultiCluster`. The rest of the licenses needed are enabled by

### Configuration

The `lsf.cluster.cluster_name` file contains the following configuration:

```
Begin Parameters
PRODUCTS= LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation LSF_Make
End Parameters

Begin Host
HOSTNAME model type server r1m mem swp RESOURCES
hostA DEFAULT LINUX86 1 - () () ()
hostB DEFAULT LINUX86 1 - () () (LSF_Base LSF_Manager LSF_MultiCluster)
hostC DEFAULT LINUX86 1 - () () ()
End Host
```

### Cluster startup

At cluster startup, all hosts are running, and the `lshosts -l` command displays the following license distribution:

```
lshosts -l

HOST_NAME: hostA
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2

RESOURCES: Not defined
RUN_WINDOWS: (always open)

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation LSF_Make)

LOAD_THRESHOLDS:
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -

HOST_NAME: hostB
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2

RESOURCES: (LSF_Base LSF_Manager LSF_MultiCluster)
```

## Partial Licensing

```
RUN_WINDOWS: (always open)
```

```
LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation)
```

```
LOAD_THRESHOLDS:
```

```
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -
```

```
HOST_NAME: hostC
```

```
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2
```

```
RESOURCES: Not defined
```

```
RUN_WINDOWS: (always open)
```

```
LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation)
```

```
LOAD_THRESHOLDS:
```

```
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -
```

All hosts are licensed for the appropriate products except `hostC`, which does not have Platform Make because its license is already being used by `hostA`. However, `hostC` is still available to run LSF jobs.

**Master host failover** If `HostA` becomes unavailable, `HostB` becomes master host. Now the `lshosts -l` command displays the following license distribution:

```
lshosts -l
```

```
HOST_NAME: hostA
```

```
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2
```

```
RESOURCES: Not defined
```

```
RUN_WINDOWS: (always open)
```

```
LICENSES_ENABLED: (LSF_Client)
```

```
LOAD_THRESHOLDS:
```

```
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -
```

```
HOST_NAME: hostB
```

```
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2
```

```
RESOURCES: (LSF_Base LSF_Manager)
```

```
RUN_WINDOWS: (always open)
```

```
LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_Sched_Fairshare LSF_Sched_Parallel
LSF_Sched_Preemption LSF_Sched_Resource_Reservation)
```

```
LOAD_THRESHOLDS:
```

```
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -
```

```

HOST_NAME: hostC
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 DEFAULT 116.1 2 1 2016M 1983M 72917M 0 Yes 1 1 2

```

RESOURCES: Not defined

RUN\_WINDOWS: (always open)

LICENSES\_ENABLED: (LSF\_Base LSF\_Manager LSF\_Sched\_Fairshare LSF\_Sched\_Parallel  
LSF\_Sched\_Preemption LSF\_Sched\_Resource\_Reservation LSF\_Make)

LOAD\_THRESHOLDS:

```

 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -

```

**Note that hostC has now picked up the available Platform Make license that was originally held by hostA.**

## Floating Client Licenses

LSF floating client is valuable if you have a cluster in which not all of the hosts will be active at the same time. In this section, you will learn how to save money through distributing your licenses efficiently.

An LSF floating client license is a type of LSF license to be shared among several client hosts at different times. Floating client licenses are not tied to specific hosts. They are assigned dynamically to any host that submits a request to LSF. The number of licenses acts as a license pool for the cluster from which LSF clients can draw required licenses. Although floating client licenses are supported, LSF does not support floating server licenses.

### Client hosts and floating client hosts

Read this to differentiate between a regular client host and a floating client host.

In LSF, you can have both client hosts and floating client hosts. The difference is in the type of license purchased.

If you purchased a regular (fixed) client license, LSF client hosts are static. The client hosts must be listed in `lsf.cluster.cluster_name`. The license is fixed to the hosts specified in `lsf.cluster.cluster_name` and whenever client hosts change, you must update it with the new host list.

If you purchased a floating client license, LSF floating client hosts are dynamic. They are not listed in `lsf.cluster.cluster_name`. Since LSF does not take into account the host name but the number of floating licenses, clients can change dynamically and licenses will be distributed to clients that request to use LSF. When you submit a job from any unlicensed host, and if there are any floating licenses free, the host will check out a license and submit your job to LSF. However, once a host checks out a floating client license, it keeps that license for the rest of the day, until midnight. A host that becomes a floating client behaves like a fixed client all day, then at 12 midnight it releases the license. At that time, the host turns back into a normal, unlicensed host, and the floating client license becomes available to any other host that needs it.

### How floating licenses work in LSF

Read this to understand how floating licenses work. You will want to read this before configuring your cluster to use this distribution technique.

When the master LIM starts up, it verifies how many floating licenses there are for the cluster as specified in `lsf.cluster.cluster_name` with the parameter `FLOAT_CLIENTS`. The master LIM checks out the licenses and keeps track of license information—how many floating licenses have been assigned, and which client hosts are using the licenses.

Floating client licenses expire at midnight (local time) on the day the license was issued. The master LIM checks the host list and removes any floating client hosts whose license has expired.

### License reset

Whenever the master LIM is restarted, all LSF floating client licenses are released and checked out again.



## Administration commands

Since LSF floating client hosts are not listed in `lsf.cluster.cluster_name`, some administration commands will not work if issued from LSF floating client hosts. Always run administration commands from server hosts.

## Floating client hosts and host types/models

This differentiates between client hosts and floating client hosts in terms of the restrictions on host types or models.

For LSF client hosts, you can list the host type and model in `lsf.cluster.cluster_name` and by default, restrict running applications on different host types.

For floating client hosts, host types and models are not included in the client information. By default, any job submissions made from floating client hosts are allowed dispatch to any host type or model.

In the same way as client and server hosts, you can specify a specific model or type when you submit a job from a floating client host.

For example:

```
bsub sleep
```

The command above is interpreted as:

- ◆ `-R "type==local"` on a client host
- ◆ `-R "type==any"` on a floating client host

## Install LSF floating client licenses

If you believe that the use of floating client licenses is appropriate for your needs, follow this procedure to install LSF floating client licenses.

### 1 Obtain the floating client license.

This is similar to getting any other license. See [Getting a permanent license](#) on page 145.

### 2 Update your license file. Add the appropriate license line in the license file `license.dat`. The LSF license must contain FEATURE lines for `LSF_Float_Client`.

Although LSF Floating Client requires a license, `LSF_Float_Client` does not appear in the `PRODUCTS` line. `LSF_Float_Client` also cannot be added as a resource for specific hosts already defined in `lsf.cluster.cluster_name`. Should these lines be present, they are ignored by LSF.

### 3 Define server hosts in `lsf.conf`.

As with any client host, specify the parameter `LSF_SERVER_HOSTS` in `lsf.conf` to define LSF server hosts for the LSF client hosts to contact.

### 4 Edit `lsf.cluster.cluster_name` by adding the `FLOAT_CLIENTS` parameter in the Parameters section:

```
...
Begin Parameters
PRODUCTS= LSF_Base LSF_Manager LSF_Sched_Fairshare
LSF_Sched_Parallel LSF_Sched_Preemption
LSF_Sched_Resource_Reservation LSF_Make
```

```
FLOAT_CLIENTS= 25
End Parameters
...
```

The `FLOAT_CLIENTS` parameter sets the size of your license pool in the cluster. When the master LIM starts up, the number of licenses specified in `FLOAT_CLIENTS` (or fewer) can be checked out for use as floating client licenses.

If the parameter `FLOAT_CLIENTS` is not specified in `lsf.cluster.cluster_name`, or there is an error in either `license.dat` or in `lsf.cluster.cluster_name`, the floating LSF client license feature is disabled.

### 5 Start the license server daemon.

See [Start the license daemons](#) on page 151.

### 6 To allow your changes to take effect, reconfigure the cluster with the commands:

- ❖ `lsadmin reconfig`
- ❖ `badmin reconfig`

---

**CAUTION:** When the LSF floating client license feature is enabled, any host can submit jobs to the cluster. You can limit which hosts can be LSF floating clients. See [Security issues with floating client licenses](#) on page 162.

---

## Security issues with floating client licenses

If you want to install or have installed floating client licenses, it is important that you read this section to inform yourself of the security issues. There are measures to compensate for these security issues (see [Configuring security for LSF floating client licenses](#) on page 162).

With LSF client licenses, when you list client hosts in `lsf.cluster.cluster_name`, there is a level of security defined since you specify the exact hosts that will be used by the LSF system. Host authentication is done in this way.

With LSF floating client licenses, you should be aware of the security issues:

- ◆ Hosts that are not specified in `lsf.cluster.cluster_name` can submit requests. This means any host can submit requests.
- ◆ Remote machines make it easier for users to submit commands with a fake user ID. As a result, if an authorized user uses the user ID `lsfadmin`, the user will be able to run commands that affect the entire cluster or shut it down and cause problems in the LSF system.

## Configuring security for LSF floating client licenses

Read this section to learn how to configure security against the issues presented in [Security issues with floating client licenses](#) on page 162.

To resolve these security issues, the LSF administrator can limit which client hosts submit requests in the cluster by adding a domain or a range of domains in `lsf.cluster.cluster_name` with the parameter `FLOAT_CLIENTS_ADDR_RANGE`.

## FLOAT\_CLIENTS\_ADDR\_RANGE parameter

This optional parameter specifies an IP address or range of addresses of domains from which floating client hosts can submit requests. Multiple ranges can be defined, separated by spaces. The IP address can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. LSF supports both formats; you do not have to map IPv4 addresses to an IPv6 format.

If the value of this parameter is undefined, there is no security and any host can be an LSF floating client.

If a value is defined, security is enabled. When this parameter is defined, client hosts that do not belong to the domain will be denied access. However, if there is an error in the configuration of this variable, by default, no host will be allowed to be an LSF floating client.

If a requesting host belongs to an IP address that falls in the specified range, the host will be accepted to become an LSF floating client.

Address ranges are validated at configuration time so they must conform to the required format. If any address range is not in the correct format, no host will be accepted as an LSF floating client and a error message will be logged in the LIM log.

## Conventions

- ◆ IP addresses are separated by spaces, and considered "OR" alternatives.
- ◆ The \* character indicates any value is allowed.
- ◆ The - character indicates an explicit range of values. For example 1-4 indicates 1,2,3,4 are allowed.
- ◆ Open ranges such as \*-30, or 10-\*, are allowed.
- ◆ If a range is specified with less fields than an IP address such as 10.161, it is considered as 10.161.\*.\*.
- ◆ This parameter is limited to 2048 characters.

## Examples

```
FLOAT_CLIENTS_ADDR_RANGE=100
```

All IPv4 and IPv6 hosts with a domain address starting with 100 will be allowed access.

- ◆ To specify only IPv4 hosts, set the value to 100.\*
- ◆ To specify only IPv6 hosts, set the value to 100:\*

```
FLOAT_CLIENTS_ADDR_RANGE=100-110.34.1-10.4-56
```

All client hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access.

Example: 100.34.9.45, 100.34.1.4, 102.34.3.20, etc. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE=100.172.1.13 100.*.30-54 124.24-*.1.*-34
```

All client hosts belonging to a domain with the address 100.172.1.13 will be allowed access. All client hosts belonging to domains starting with 100, then any number, then a range of 30 to 54 will be allowed access. All client hosts belonging to domains starting with 124, then from 24 onward, then 1, then from 0 to 34 will be allowed access. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE=12.23.45.*
```

All client hosts belonging to domains starting with 12.23.45 are allowed. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE=100.*43
```

The \* character can only be used to indicate any value. In this example, an error will be inserted in the LIM log and no hosts will be accepted to become LSF floating clients. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE=100.*43 100.172.1.13
```

Although one correct address range is specified, because \*43 is not correct format, the entire line is considered not valid. An error will be inserted in the LIM log and no hosts will be accepted to become LSF floating clients. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE = 3ffe
```

All client IPv6 hosts with a domain address starting with 3ffe will be allowed access. No IPv4 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE = 3ffe:ffe::88bb:*
```

Expands to 3ffe:ffe:0:0:0:88bb:\*. All IPv6 client hosts belonging to domains starting with 3ffe:ffe::88bb:\* are allowed. No IPv4 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE = 3ffe-4fff:ffe::88bb:aa-ff 12.23.45.*
```

All IPv6 client hosts belonging to domains starting with 3ffe up to 4fff, then ffe::88bb, and ending with aa up to ff are allowed. All IPv4 client hosts belonging to domains starting with 12.23.45 are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE = 3ffe-*:ffe::88bb:*-ff
```

All IPv6 client hosts belonging to domains starting with 3ffe up to ffff and ending with 0 up to ff are allowed. No IPv4 hosts are allowed.

## Checking that security is enabled

Take this step after you have configured security. You are shown how to check that security has been configured properly.

After you configure `FLOAT_CLIENTS_ADDR_RANGE`, check the master LIM log file on the LSF master host (`LSF_LOGDIR/lim.log.master_host_name`) to make sure this parameter is correctly set. If this parameter is not set or is wrong, this will be indicated in the log file.

## Verify LSF floating client license is working

Perform this procedure after setting up your floating client license to verify that your floating client license is enabled.

- 
- 1 Start a cluster.
  - 2 Run the `lshosts` command from a host listed in `lsf.cluster.cluster_name`:  

```
lshosts
```

In the following example, only `hostA` and `hostB` are defined in `lsf.cluster.cluster_name`. `hostA` is a server and master host, and `hostB` is a static client. If you type the command from `hostA` or `hostB`, you will get the following output:

```
lshosts
HOST_NAME type model cpuf ncpus maxmem maxswp server RESOURCES
hostA SUNSOL DEFAULT 1.0 1 128M 602M Yes ()
hostB SUNSOL DEFAULT 1.0 - - - No ()
```

**3** Submit a job from a host not listed in `lsf.cluster.cluster_name`.

For example, if you submitted the following job from `hostC`:

```
bsub sleep 1000
```

You would get the following response:

```
Job <104> is submitted to default queue <normal>.
```

**4** From any LSF host, with `LSF_ENVDIR` set to this cluster, run the `lshosts` command:

```
lshosts
HOST_NAME type model cpuf ncpus maxmem maxswp server RESOURCES
hostA SUNSOL DEFAULT 1.0 1 128M 602M Yes ()
hostB SUNSOL DEFAULT 1.0 - - - No ()
hostC UNKNOWN UNKNOWN 1.0 - - - No ()
```

In the above example, although `hostC` shows the type `UNKNOWN` and `hostA` and `hostB` are of type `SUNSOL` (Sun Solaris), the job will be allowed to be executed on any host type because `hostC` is a floating client host without any model or type restrictions specified at job submission.

**5** From any host, run the `lshosts -l` command:

```
lshosts -l hostC
```

where `hostC` is a floating client host.

```
HOST_NAME: hostC
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server
UNKNOWN UNKNOWN 1.0 - - - - - - No
```

```
RESOURCES: Not defined
```

```
RUN_WINDOWS: Not applicable for client-only host
```

```
LICENSES_ENABLED: (LSF_Float_Client)
```

## Troubleshooting License Issues

- ◆ ["lsadmin reconfig" gives "User permission denied" message](#) on page 166
- ◆ [Primary cluster administrator receives email "Your cluster has experienced license overuse" message](#) on page 166
- ◆ [lsadmin command fails with "ls\\_gethostinfo: Host does not have a software license" on page 166](#)
- ◆ [LSF commands give "Host does not have a software license" on page 167](#)
- ◆ [LSF commands fail with "ls\\_initdebug: Unable to open file lsf.conf" on page 167](#)
- ◆ [lmgrd fails with message "Port already in use" on page 168](#)

### "lsadmin reconfig" gives "User permission denied" message

If you ran `lsfinstall` as a non-root user to install a multi-user cluster, the LSF administration commands `lsadmin` and `badadmin` might give the error message "User permission denied".

Use the following commands to change the ownership for `lsadmin` and `badadmin` to root and the file permission mode to `-rwsr-xr-x`:

```
chown root lsadmin badadmin
chmod 4755 lsadmin badadmin
```

Now the user ID bit for the owner is `setuid`. If `lsadmin` and `badadmin` are in a directory shared through NFS, the directory must be shared and mounted with `setuid` enabled. Do not mount with the `nosuid` flag. If your site does not permit this, copy `lsadmin` and `badadmin` to `/usr/bin` or `/bin`.

### Primary cluster administrator receives email "Your cluster has experienced license overuse" message

This occurs when your cluster is using more licenses than you have purchased. LSF allows for some overuse due to the peak usage of the cluster.

See the `lsf.cluster_name.license.acct` file for details of the peak license usage of your cluster:

**OK** Peak usage is less than the maximum license availability

**OVERUSE** Peak usage is more than the maximum license availability

If your cluster experiences frequent license violations or overuse, contact Platform Computing or your Platform LSF vendor to get more licenses, or plan your cluster to reduce the license usage during peak periods.

### lsadmin command fails with "ls\_gethostinfo: Host does not have a software license"

This may occur when you have installed the new key but have an old (unlicensed) LIM running on the LSF master.

---

1 On the LSF master, enter the command:

```
ps -ef | grep lim
```

- 2 Kill the LIM, using one of the following commands:
 

```
kill lim_PID
kill -9 lim_PID
```
- 3 After the old LIM has died, start the new LIM on the master host using one of the following methods:
  - ❖ `lsadmin limstartup`
  - ❖ `LSF_SERVERDIR/lim` as root.

## LSF commands give "Host does not have a software license"

You may see this message after running `lsid`, `lshosts`, or other `ls*` commands.

Typical problems: and their solutions:

| If you experience this problem ...                                                                                                                                                                                                                                    | Do the following:                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Your demo license (not tied to FLEXlm server) has expired.                                                                                                                                                                                                            | Check the <code>license.dat</code> file to check the expiry date. If your license has expired, contact your account manager to obtain a new demo key or a permanent license.                                                                                                      |
| Your license file may have incorrect formatting. One of the following things may be responsible:<br>The license file was edited in Windows and incorrect line ending characters (^M) exist in the file.<br>The license file may have more than one FEATURE on a line. | You should have a line break at the end of each FEATURE line. Remove the excess ^M characters or line breaks from the license file.<br>If the license key is tied to a FLEXlm server, restart <code>lmgrd</code> .<br>Restart the master LIM.                                     |
| The LSF master host is unable to communicate with the FLEXlm server.                                                                                                                                                                                                  | Check the network communication by entering the command:<br><code>ping FLEXlm_server</code>                                                                                                                                                                                       |
| License daemons ( <code>lmgrd</code> and <code>lsf_ld</code> ) are not running on the FLEXlm server.                                                                                                                                                                  | Check if <code>lmgrd</code> and <code>lsf_ld</code> are running by typing:<br><code>ps -ef   egrep 'lmgrd lsf_ld'</code><br>on the FLEXlm server. If not:<br>Check the <code>license.log</code> file for error messages.<br>Start <code>lmgrd</code> .<br>Restart the master LIM. |

## LSF commands fail with "ls\_initdebug: Unable to open file lsf.conf"

You might see this message after running `lsid`. This message indicates that the LSF commands cannot access the `lsf.conf` file or `lsf.conf` does not exist in `LSF_ENVDIR`.

Solution:

- ◆ Use `LSF_CONFDIR/csrhc.lsf` or `LSF_CONFDIR/profile.lsf` to set up your LSF environment, or
- ◆ If you know the location of `lsf.conf`, set the `LSF_ENVDIR` environment variable to point to the directory containing the `lsf.conf` file.

## lmgrd fails with message "Port already in use"

The port number defined in `LSF_LICENSE_FILE` and `license.dat` is being used by another application (by default, LSF uses port number 1700).

Possible causes:

| If you experience this problem ...                                       | Do the following:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lmgrd is already running for this license                                | Use <code>ps -ef</code> and make sure that <code>lmgrd</code> and <code>lsf_lid</code> are not running.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| lmgrd has been stopped and the operating system has not cleared the port | Wait a few minutes for the OS to clear this port.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Another process is using the same port (this is not likely)              | <p>If the port number is being used by another application, execute the following to change the port number used by LSF:</p> <ol style="list-style-type: none"> <li>1 Edit <code>license.dat</code> and change the port number in the line:           <pre>SERVER flexlm_server 3f8b6a3 1700</pre> <p>The fourth field on the <code>SERVER</code> line of <code>license.dat</code> specifies the TCP port number that the FLEXlm server uses. Choose an unused port number. The default port set by FLEXlm is 1700. Platform LSF usually uses port numbers in the range 3879 to 3882, so the numbers from 3883 forward are good alternate choices.</p> </li> <li>2 In <code>lsf.conf</code>:           <ul style="list-style-type: none"> <li>❖ If <code>LSF_LICENSE_FILE</code> is defined as follows:               <pre>LSF_LICENSE_FILE=port_number@flexlm_server</pre>               (for example: <code>1700@hostA</code>), the port number must be changed accordingly.             </li> <li>❖ If <code>LSF_LICENSE_FILE</code> points to the license file path (for example:               <pre>LSF_LICENSE_FILE=/usr/local/lsf/conf/license.dat</pre>), no changes are required.             </li> </ul> </li> <li>a Restart <code>lmgrd</code>.</li> </ol> |



## Managing LSF on Platform EGO

### Contents

- ◆ [About LSF on Platform EGO](#) on page 170
- ◆ [LSF and EGO directory structure](#) on page 173
- ◆ [Configuring LSF and EGO](#) on page 177
- ◆ [Managing LSF daemons through EGO](#) on page 181
- ◆ [EGO Administrative Basics](#) on page 184
- ◆ [Logging and troubleshooting](#) on page 201
- ◆ [Frequently asked questions](#) on page 209

## About LSF on Platform EGO

LSF on Platform EGO allows EGO to serve as the central resource broker, enabling enterprise applications to benefit from sharing of resources across the enterprise grid.

- ◆ *Scalability*—EGO enhances LSF scalability. Currently, the LSF scheduler has to deal with a large number of jobs. EGO provides management functionality for multiple schedulers that co-exist in one EGO environment. In LSF Version 7, although only a single instance of LSF is available on EGO, the foundation is established for greater scalability in follow-on releases that will allow multiple instances of LSF on EGO.
- ◆ *Robustness*—In previous releases, LSF functioned as both scheduler and resource manager. EGO decouples these functions, making the entire system more robust. EGO reduces or eliminates downtime for LSF users while resources are added or removed.
- ◆ *Reliability*—In situations where service is degraded due to noncritical failures such as sbatchd or RES, by default, LSF does not automatically restart the daemons. The EGO Service Controller can monitor all LSF daemons and automatically restart them if they fail. Similarly, the EGO Service Controller can also monitor and restart other critical processes such as FlexLM and lmgrd.
- ◆ *Additional scheduling functionality*—EGO provides the foundation for EGO-enabled SLA, which provides LSF with additional and important scheduling functionality.
- ◆ *Centralized management and administration framework.*
- ◆ *Single reporting framework*—across various application heads built around EGO.

## What is Platform EGO?

Platform Enterprise Grid Orchestrator (EGO) allows developers, administrators, and users to treat a collection of distributed software and hardware resources on a shared computing infrastructure (cluster) as parts of a single virtual computer.

EGO assesses the demands of competing business services (consumers) operating within a cluster and dynamically allocates resources so as to best meet a company's overriding business objectives. These objectives might include

- ◆ Reducing the time or the cost of providing key business services
- ◆ Maximizing the revenue generated by existing computing infrastructure
- ◆ Configuring, enforcing, and auditing service plans for multiple consumers
- ◆ Ensuring high availability and business continuity through disaster scenarios
- ◆ Simplifying IT management and reducing management costs
- ◆ Consolidating divergent and mixed computing resources into a single virtual infrastructure that can be shared transparently between many business users

Platform EGO also provides a full suite of services to support and manage resource orchestration. These include cluster management, configuration and auditing of service-level plans, resource facilitation to provide fail-over if a master host goes down, monitoring and data distribution.

EGO is only sensitive to the *resource requirements* of business services; EGO has no knowledge of any run-time dynamic parameters that exist for them. This means that EGO does not interfere with how a business service chooses to use the resources it has been allocated.

## How does Platform EGO work?

Platform products work in various ways to match business service (consumer) demands for resources with an available supply of resources. While a specific clustered application manager or consumer (for example, an LSF cluster) identifies what its resource demands are, Platform EGO is responsible for supplying those resources. Platform EGO determines the number of resources each consumer is entitled to, takes into account a consumer's priority and overall objectives, and then allocates the number of required resources (for example, the number of slots, virtual machines, or physical machines).

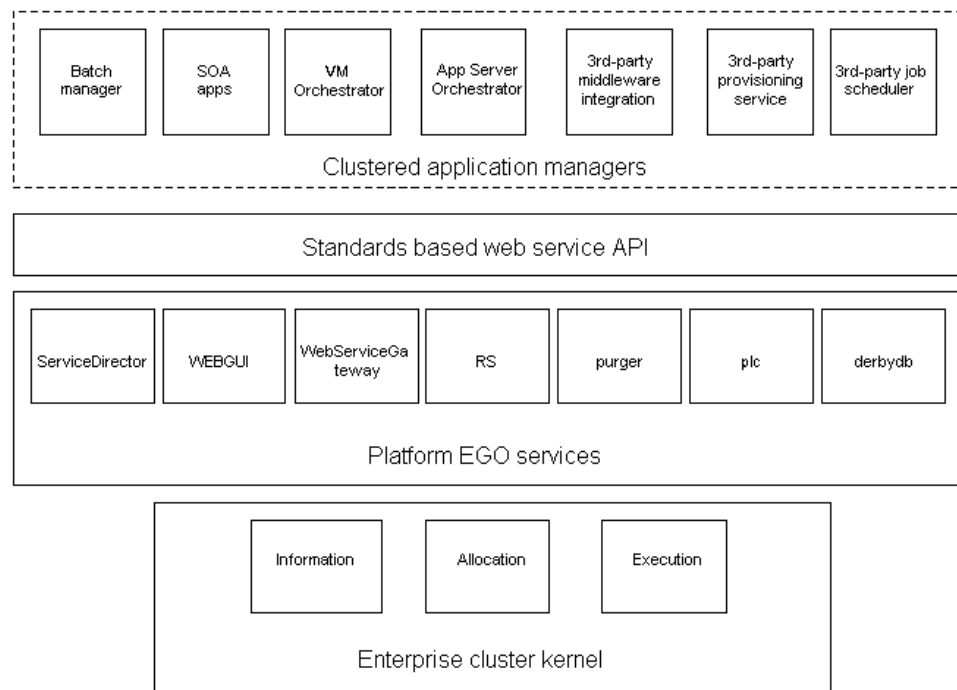
Once the consumer receives its allotted resources from Platform EGO, the consumer applies its own rules and policies. How the consumer decides to balance its workload across the fixed resources allotted to it is not the responsibility of EGO.

So how does Platform EGO know the demand? Administrators or developers use various EGO interfaces (such as the SDK or CLI) to tell EGO what constitutes a demand for more resources. When Platform LSF identifies that there is a demand, it then distributes the required resources based on the resource plans given to it by the administrator or developer.

For all of this to happen smoothly, various components are built into Platform EGO. Each EGO component performs a specific job.

## Platform EGO components

Platform EGO comprises a collection of cluster orchestration software components. The following figure shows overall architecture and how these components fit within a larger system installation and interact with each other:



## Key EGO concepts

- Consumers** A consumer represents an entity that can demand resources from the cluster. A consumer might be a business service, a business process that is a complex collection of business services, an individual user, or an entire line of business.
- EGO resources** Resources are physical and logical entities that can be requested by a client. For example, an application (client) requests a processor (resource) in order to run. Resources also have attributes. For example, a host has attributes of memory, processor utilization, operating systems type, etc.
- Resource distribution tree** The resource distribution tree identifies consumers of the cluster resources, and organizes them into a manageable structure.
- Resource groups** Resource groups are logical groups of hosts. Resource groups provide a simple way of organizing and grouping resources (hosts) for convenience; instead of creating policies for individual resources, you can create and apply them to an entire group. Groups can be made of resources that satisfy a specific requirement in terms of OS, memory, swap space, CPU factor and so on, or that are explicitly listed by name.
- Resource distribution plans** The resource distribution plan, or resource plan, defines how cluster resources are distributed among consumers. The plan takes into account the differences between consumers and their needs, resource properties, and various other policies concerning consumer rank and the allocation of resources.
- The distribution priority is to satisfy each consumer's reserved ownership, then distribute remaining resources to consumers that have demand.
- Services** A service is a self-contained, continuously running process that accepts one or more requests and returns one or more responses. Services may have multiple concurrent service instances running on multiple hosts. All Platform EGO services are automatically enabled by default at installation.
- Run `egosh` or use the Platform Management Console (PMC) to check service status.
- If EGO is disabled, the `egosh` command cannot find `ego.conf` or cannot contact `vemkd` (not started), and the following message is displayed:
- ```
You cannot run the egosh command because the administrator has chosen not to enable EGO in lsf.conf: LSF_ENABLE_EGO=N.
```
- EGO user accounts** A user account is a Platform system user who can be assigned to any role for any consumer in the tree. User accounts include optional contact information, a name, and a password.
- EGO user roles** There are three user roles to which any user account can be assigned. Each user role has a specific level of accessibility and control in the PMC.
- Roles include Cluster Administrator, Consumer Administrator, and Consumer User.

LSF and EGO directory structure

The following tables describe the purpose of each sub-directory and whether they are writable or non-writable by LSF.

LSF_TOP

Directory Path	Description	Attribute
LSF_TOP/7.0	LSF 7.0 binaries and other machine dependent files	Non-writable
LSF_TOP/conf	LSF 7.0 configuration files You must be LSF administrator or root to edit files in this directory	Writable by the LSF administrator, master host, and master candidate hosts
LSF_TOP/log	LSF 7.0 log files	Writable by all hosts in the cluster
LSF_TOP/work	LSF 7.0 working directory	Writable by the master host and master candidate hosts, and is accessible to slave hosts

EGO, GUI, and PERF directories

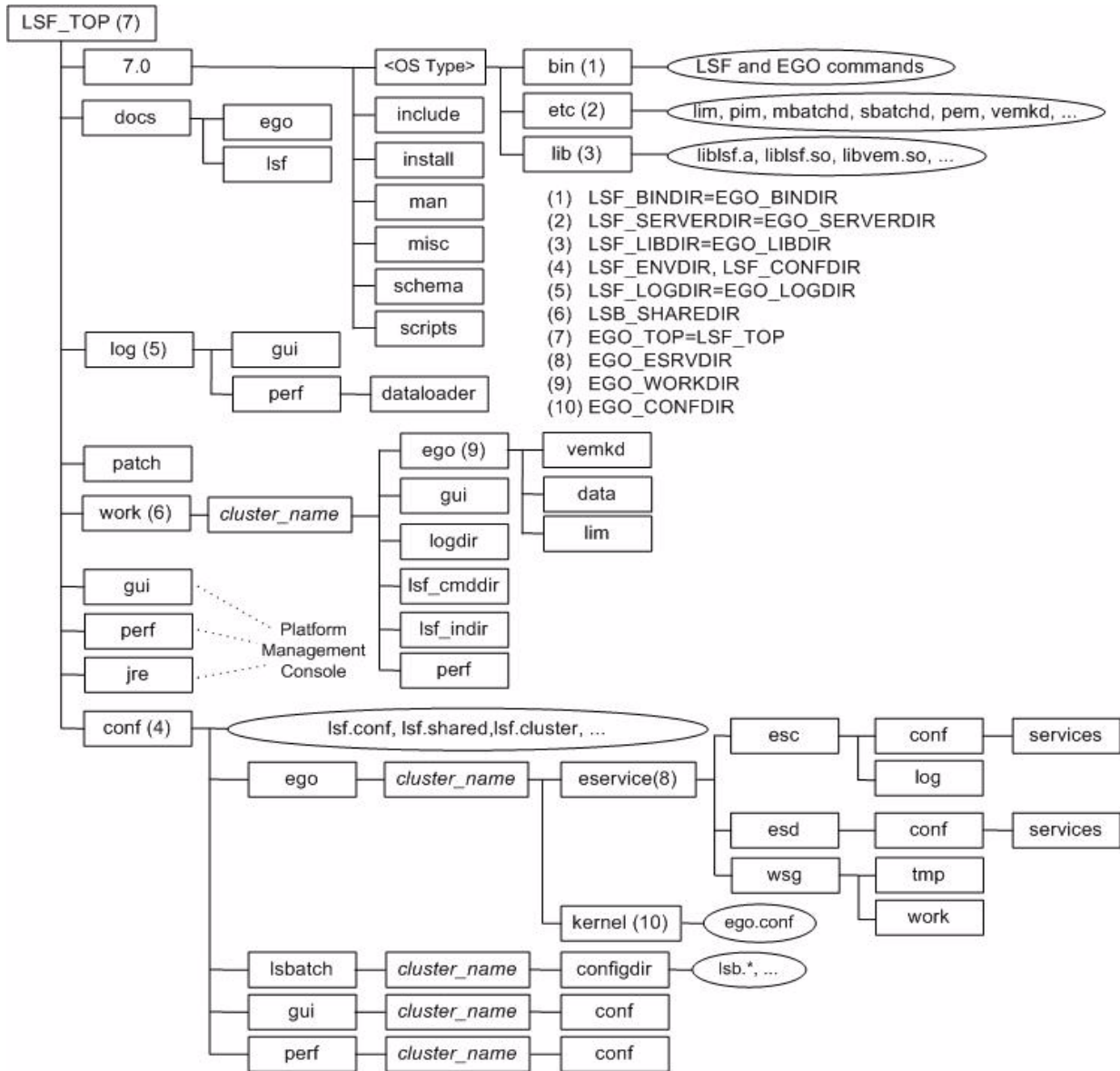
Directory Path	Description	Attribute
LSF_BINDIR	EGO binaries and other machine dependent files	Non-writable
LSF_CONFDIR/ego/ <i>cluster_name</i> /eservice (EGO_ESRVDIR)	EGO services configuration and log files.	Writable
LSF_CONFDIR/ego/ <i>cluster_name</i> /kernel (EGO_CONFDIR, LSF_EGO_ENVDIR)	EGO kernel configuration, log files and working directory, including conf/log/work	Writable
LSB_SHAREDIR/ <i>cluster_name</i> /ego (EGO_WORKDIR)	EGO working directory	Writable
LSF_TOP/perf/1.2	PERF commands, library and schema	Non-writable
LSF_CONFDIR/perf/ <i>cluster_name</i> /conf (PERF_CONFDIR)	PERF configuration	Writable
LSB_SHAREDIR/ <i>cluster_name</i> /perf/data (PERF_DATADIR)	PERF embedded data files for derby	Writable
LSF_TOP/perf/1.2/etc	PERF script command for services	Non-writable
LSF_TOP/log/perf (PERF_LOGDIR)	PERF log files	Writable
LSB_SHAREDIR/ <i>cluster_name</i> /perf (PERF_WORKDIR)	PERF working directory	Writable
LSF_TOP/jre	Java Runtime Environment	Non-writable
LSF_TOP/gui	GUI	Non-writable
LSF_CONFDIR/gui/ <i>cluster_name</i> /conf (GUI_CONFDIR)	GUI configuration	Writable
LSB_SHAREDIR/ <i>cluster_name</i> /gui (CATALINA_WORKDIR, CATALINA_TMPDIR)	GUI working directory	Writable
LSF_TOP/log/gui (GUI_LOGDIR)	GUI log files	Writable
LSF_TOP/gui/1.2/	GUI binaries and tomcat	Non-writable
LSF_TOP/gui/1.2/tomcat	Tomcat web server	Writable

NOTE: Several directories under LSF_TOP/gui/1.2/tomcat are writable by Tomcat servers. You should install the whole Tomcat directory on a writable file system.

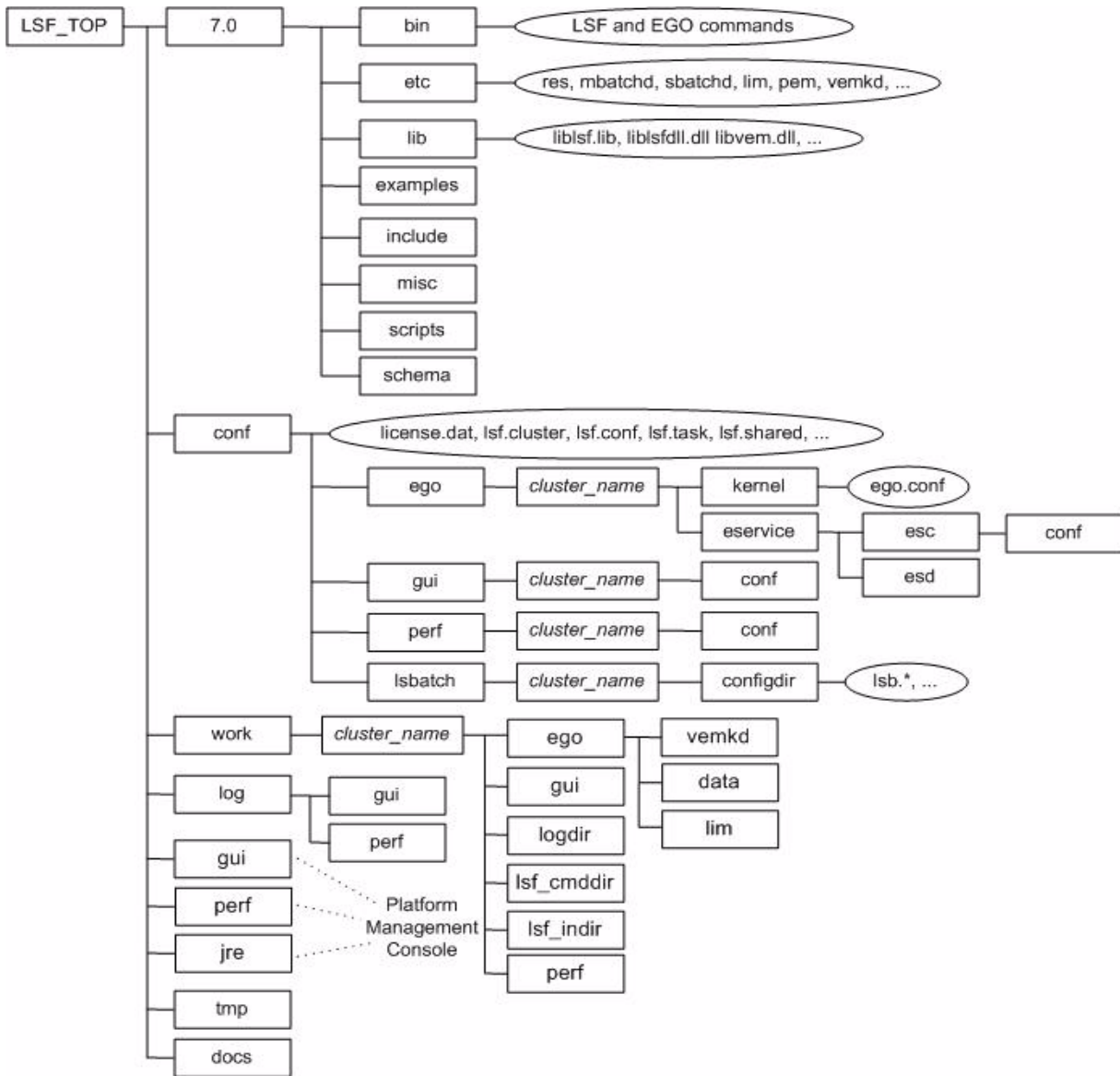
Example directory structures

UNIX and Linux

The following figures show typical directory structures for a new UNIX or Linux installation with `lsfinstall`. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Microsoft Windows The following diagram shows an example directory structure for a Windows installation.



Configuring LSF and EGO

EGO configuration files for LSF daemon management (res.xml and sbatchd.xml)

The following files are located in `EGO_ESRVDIR/esc/conf/services/`:

- ◆ `res.xml`—EGO service configuration file for `res`.
- ◆ `sbatchd.xml`—EGO service configuration file for `sbatchd`.

When LSF daemon control through EGO Service Controller is configured, `lsadmin` uses the reserved EGO service name `res` to control the LSF `res` daemon, and `badmin` uses the reserved EGO service name `sbatchd` to control the LSF `sbatchd` daemon.

How to handle parameters in `lsf.conf` with corresponding parameters in `ego.conf`

When EGO is enabled, existing LSF parameters (parameter names beginning with `LSB_` or `LSF_`) that are set only in `lsf.conf` operate as usual because LSF daemons and commands read both `lsf.conf` and `ego.conf`.

Some existing LSF parameters have corresponding EGO parameter names in `ego.conf` (`LSF_CONFDIR/lsf.conf` is a separate file from `LSF_CONFDIR/ego/cluster_name/kernel/ego.conf`). You can keep your existing LSF parameters in `lsf.conf`, or you can set the corresponding EGO parameters in `ego.conf` that have not already been set in `lsf.conf`.

You cannot set LSF parameters in `ego.conf`, but you can set the following EGO parameters related to LIM, PIM, and ELIM in either `lsf.conf` or `ego.conf`:

- ◆ `EGO_DAEMONS_CPUS`
- ◆ `EGO_DEFINE_NCPUS`
- ◆ `EGO_SLAVE_CTRL_REMOTE_HOST`
- ◆ `EGO_WORKDIR`
- ◆ `EGO_PIM_SWAP_REPORT`

You cannot set any other EGO parameters (parameter names beginning with `EGO_`) in `lsf.conf`. If EGO is not enabled, you can only set these parameters in `lsf.conf`.

NOTE: If you specify a parameter in `lsf.conf` and you also specify the corresponding parameter in `ego.conf`, the parameter value in `ego.conf` takes precedence over the conflicting parameter in `lsf.conf`.

If the parameter is not set in either `lsf.conf` or `ego.conf`, the default takes effect depends on whether EGO is enabled. If EGO is not enabled, then the LSF default takes effect. If EGO is enabled, the EGO default takes effect. In most cases, the default is the same.

Some parameters in `lsf.conf` do not have exactly the same behaviour, valid values, syntax, or default value as the corresponding parameter in `ego.conf`, so in general, you should not set them in both files. If you need LSF parameters for backwards compatibility, you should set them only in `lsf.conf`.

If you have LSF 6.2 hosts in your cluster, they can only read `lsf.conf`, so you must set LSF parameters only in `lsf.conf`.

LSF and EGO corresponding parameters

The following table summarizes existing LSF parameters that have corresponding EGO parameter names. You must continue to set other LSF parameters in `lsf.conf`.

lsf.conf parameter	ego.conf parameter
LSF_API_CONNTIMEOUT	EGO_LIM_CONNTIMEOUT
LSF_API_RECVTIMEOUT	EGO_LIM_RECVTIMEOUT
LSF_CLUSTER_ID (Windows)	EGO_CLUSTER_ID (Windows)
LSF_CONF_RETRY_INT	EGO_CONF_RETRY_INT
LSF_CONF_RETRY_MAX	EGO_CONF_RETRY_MAX
LSF_DEBUG_LIM	EGO_DEBUG_LIM
LSF_DHPC_ENV	EGO_DHPC_ENV
LSF_DYNAMIC_HOST_TIMEOUT	EGO_DYNAMIC_HOST_TIMEOUT
LSF_DYNAMIC_HOST_WAIT_TIME	EGO_DYNAMIC_HOST_WAIT_TIME
LSF_ENABLE_DUALCORE	EGO_ENABLE_DUALCORE
LSF_GET_CONF	EGO_GET_CONF
LSF_GETCONF_MAX	EGO_GETCONF_MAX
LSF_LIM_DEBUG	EGO_LIM_DEBUG
LSF_LIM_PORT	EGO_LIM_PORT
LSF_LOCAL_RESOURCES	EGO_LOCAL_RESOURCES
LSF_LOG_MASK	EGO_LOG_MASK
LSF_MASTER_LIST	EGO_MASTER_LIST
LSF_PIM_INFODIR	EGO_PIM_INFODIR
LSF_PIM_SLEEPTIME	EGO_PIM_SLEEPTIME
LSF_PIM_SLEEPTIME_UPDATE	EGO_PIM_SLEEPTIME_UPDATE
LSF_RSH	EGO_RSH
LSF_STRIP_DOMAIN	EGO_STRIP_DOMAIN
LSF_TIME_LIM	EGO_TIME_LIM

Parameters that have changed in LSF 7

The default for `LSF_LIM_PORT` has changed to accommodate EGO default port configuration. On EGO, default ports start with `lim` at 7869, and are numbered consecutively for `pem`, `vemkd`, and `egosc`.

This is different from previous LSF releases where the default `LSF_LIM_PORT` was 6879. `res`, `sbatchd`, and `mbatchd` continue to use the default pre-version 7 ports 6878, 6881, and 6882.

Upgrade installation preserves any existing port settings for `lim`, `res`, `sbatchd`, and `mbatchd`. EGO `pem`, `vemkd`, and `egosc` use default EGO ports starting at 7870, if they do not conflict with existing `lim`, `res`, `sbatchd`, and `mbatchd` ports.

EGO connection ports and base port

On every host, a set of connection ports must be free for use by LSF and EGO components.

LSF and EGO require exclusive use of certain ports for communication. EGO uses the same four consecutive ports on every host in the cluster. The first of these is called the base port.

The default EGO base connection port is 7869. By default, EGO uses four consecutive ports starting from the base port. By default, EGO uses ports 7869-7872.

The ports can be customized by customizing the base port. For example, if the base port is 6880, EGO uses ports 6880-6883.

LSF and EGO needs the same ports on every host, so you must specify the same base port on every host.

Special resource groups for LSF master hosts

By default, Platform LSF installation defines a special resource group named `ManagementHosts` for the Platform LSF master host. (In general, Platform LSF master hosts are dedicated hosts; the `ManagementHosts` EGO resource group serves this purpose.)

Platform LSF master hosts must not be subject to any lend, borrow, or reclaim policies. They must be exclusively owned by the Platform LSF consumer.

The default Platform EGO configuration is such that the `LSF_MASTER_LIST` hosts and the execution hosts are in different resource groups so that different resource plans can be applied to each group.

Disabling Platform EGO after installation

Prerequisites: Platform LSF 7 Update 2 is installed with `ENABLE_EGO=Y` (or EGO was enabled after installation).

- 1 Log on to the LSF master host as root or the primary LSF administrator.
- 2 Shut down the cluster.


```
lsfshutdown
```
- 3 Edit `lsf.conf`.
- 4 Set `LSF_ENABLE_EGO=N`.
- 5 Comment out or remove `LSF_EGO_ENVDIR`.
- 6 Set your environment using `profile.lsf` or `cshrc.lsf`.
- 7 Restart the cluster.


```
lsfstartup
```

Enabling Platform EGO after installation

Prerequisites: Platform LSF 7 Update 2 is installed with `ENABLE_EGO=N` (or EGO was disabled after installation).

- 1 Log on to the LSF master host as root or the primary LSF administrator.
- 2 Shut down the cluster.


```
lsfshutdown
```

Configuring LSF and EGO

- 3 Edit `lsf.conf`.
- 4 Set `LSF_ENABLE_EGO=Y`.
- 5 Uncomment out or set `LSF_EGO_ENVDIR`.

For example:

```
LSF_EGO_ENVDIR=/usr/share/lsf7/conf/ego/lsf702/kernel.
```

- 6 Set your environment using `profile.lsf` or `cshrc.lsf`.
 - 7 Restart the cluster.
`lsfstartup`
-

Managing LSF daemons through EGO

EGO daemons

Daemons in LSF_SERVERDIR	Description
vemkd	Started by lim on master host
pem	Started by lim on every host
egosc	Started by vemkd on master host

LSF daemons

Daemons in LSF_SERVERDIR	Description
lim	lim runs on every host. On UNIX, lim is either started by lsadmin through rsh/ssh or started through rc file. On Windows, lim is started as a Windows service.
pim	Started by lim on every host
mbatchd	Started by sbatchd on master host
mbschd	Started by mbatchd on master host
sbatchd	Under OS startup mode, sbatchd is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, sbatchd is started as a Windows service. Under EGO Service Controller mode, sbatchd is started by pem as an EGO service on every host.
res	Under OS startup mode, res is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, res is started as a Windows service. Under EGO Service Controller mode, res is started by pem as an EGO service on every host.

Operating System daemon control

Operating system startup mode is the same as previous releases:

- ◆ On UNIX, administrators configure the autostart of `sbatchd` and `res` in the operating system (`/etc/rc` file or `inittab`) and use `lsadmin` and `badmin` to start LSF daemons manually through `rsh` or `ssh`.
- ◆ On Windows, `sbatchd` and `res` are started as Windows services.

EGO Service Controller daemon control

Under EGO Service Control mode, administrators configure the EGO Service Controller to start `res` and `sbatchd`, and restart them if they fail.

You can still run `lsadmin` and `badmin` to start LSF manually, but internally, `lsadmin` and `badmin` communicates with the EGO Service Controller, which actually starts `sbatchd` and `res` as EGO services.

If EGO Service Controller management is configured and you run `badmin hshutdown` and `lsadmin resshutdown` to manually shut down LSF, the LSF daemons are not restarted automatically by EGO. You must run `lsadmin resstartup` and `badmin hstartup` to start the LSF daemons manually.

Enable LSF daemon management through EGO Service Controller

Enable LSF daemon management at installation

You should enable LSF daemon management through EGO Service Controller at installation.

To avoid conflicts with existing LSF startup scripts, leave `EGO_DAEMON_CONTROL` undefined if you use a script (for example in `/etc/rc` or `/etc/inittab`) to start LSF daemons.

- 1 Set `EGO_DAEMON_CONTROL="Y"` in `install.config`.

This sets `LSF_EGO_DAEMON_CONTROL="Y"` in `lsf.conf`, and configures the start type for the `sbatchd` and `res` EGO services to `AUTOMATIC` in `res.xml` and `sbatchd.xml` under `EGO_ESRVDIR/esc/conf/services`.

Change LSF daemon management after installation

If you did not enable LSF daemon management through EGO Service Controller at installation, complete the following steps to set LSF `res` and `sbatchd` startup to `AUTOMATIC` in the EGO configuration files `res.xml` and `sbatchd.xml` under `EGO_ESRVDIR/esc/conf/services`:

To avoid conflicts with existing LSF startup scripts, leave `EGO_DAEMON_CONTROL` undefined if you use a script (for example in `/etc/rc` or `/etc/inittab`) to start LSF daemons.

- 1 Log on to the LSF master host as `lsfadmin`.
- 2 Shut down the LSF cluster.
- 3 Edit `lsf.conf` and set `LSF_EGO_DAEMON_CONTROL="Y"`.
`LSF_EGO_DAEMON_CONTROL` enables EGO Service Controller to control LSF `res` and `sbatchd` startup.

IMPORTANT: After installation, `LSF_EGO_DAEMON_CONTROL` alone *does not* change the start type for the `sbatchd` and `res` EGO services to `AUTOMATIC` in `res.xml` and `sbatchd.xml` under `EGO_ESRVDIR/esc/conf/services`. You must edit these files and set the `<sc: StartType>` parameter to `AUTOMATIC`.

- 4 Edit `res.xml` and `sbatchd.xml` under `EGO_ESRVDIR/esc/conf/services` and change `MANUAL` to `AUTOMATIC`.
EGO Service Controller starts `res` and `sbatchd` automatically, and restarts them if they fail
- 5 Run `lsadmin reconfig` to reconfigure LIM.
- 6 Run `badmin mbdrestart` to restart `mbatchd`.

Permissions required for daemon control

To control all daemons in the cluster, you must

- ◆ Be logged on as root or as a user listed in the `/etc/lsf.sudoers` file. See the *Platform LSF Configuration Reference* for configuration details of `lsf.sudoers`.
- ◆ Be able to run the `rsh` or `ssh` commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring the `rsh` and `ssh` commands. The shell command specified by `LSF_RSH` in `lsf.conf` is used before `rsh` is tried.

Bypass EGO login at startup (`lsf.sudoers`)

Prerequisites: You must be the LSF administrator (`lsfadmin`) or root to configure `lsf.sudoers`.

When LSF daemons control through EGO Service Controller is configured, users must have EGO credentials for EGO to start `res` and `sbatchd` services. By default, `lsadmin` and `badmin` invoke the `egosh user logon` command to prompt for the user name and password of the EGO administrator to get EGO credentials.

- 1 Configure `lsf.sudoers` to bypass EGO login to start `res` and `sbatchd` automatically.

Set the following parameters:

- ◆ `LSF_EGO_ADMIN_USER`—User name of the EGO administrator. The default administrator name is `Admin`.
- ◆ `LSF_EGO_ADMIN_PASSWD`—Password of the EGO administrator.

EGO Administrative Basics

See *Administering and Using Platform EGO* for detailed information about EGO administration.

Enable the full Platform Management Console

By default, only the LSF reporting feature is enabled in the Platform Management Console (PMC) after installation. Complete the following steps to enable the full PMC functionality for managing LSF on EGO.

- 1 Download and install the Platform Management Console (PMC) from one of the distribution packages from `/distrib/7.0/platform_lsf_update2/`.

```
ftp> get platform_lsf_update2/lsf7Update2_pmc_linux-x86.tar.Z
```

OR

```
ftp> get platform_lsf_update2/lsf7Update2__pmc_linux-x86_64.tar.Z
```

NOTE: To take advantage of the Platform LSF reporting feature, you *must* download and install the Platform Management Console. The reporting feature is only supported on the same platforms as the Platform Management Console: 32-bit and 64-bit x86 Windows and Linux operating systems.

- 2 With an XML editor, open `pmc_conf_ego.xml`.
 - ❖ **Windows:** `LSF_CONFDIR\gui\cluster_name\conf\pmcconf\pmc_conf_ego.xml`
 - ❖ **Linux:** `LSF_CONFDIR/gui/cluster_name/conf/pmccconf/pmc_conf_ego.xml`

If you ran `egoconfig mghost` after installation, then `pmc_conf_ego.xml` is located in the EGO shared directory:

 - ❖ **Windows:** `shared_dir\gui\cluster_name\conf\pmcconf\pmc_conf_ego.xml`
 - ❖ **Linux:** `shared_dir/gui/cluster_name/conf/pmccconf/pmc_conf_ego.xml`
- 3 In the configuration section, locate the parameter:
`<Name>OnlyShowReport</Name>`.
- 4 In the `<Name>` parameter, change `<Value>>true</Value>` to `<Value>>false</Value>`.
- 5 Save and close `pmc_conf_ego.xml`.
- 6 Restart the WEBGUI service.

Log on to the Platform Management Console

Prerequisites: You have set your environment on this host. You are logged into the operating system as `egoadmin`.

The Platform Management Console allows you to monitor, administer, and configure your cluster.

-
- 1 If you do not already know the web server URL, run `egosh client view`.
 - a Look for the client name preceded by “GUIURL”.
For example, `GUIURL_Host_W` (this is the fully qualified host name).
 - b Look for the `DESCRIPTION` line beneath the client name to find the web server URL, and then copy it.
For example: `http://Host_W:8080/platform`.
 - 2 Launch any web browser and enter the address of the web server URL.
The format of the URL is always `http://host_name:port_number/platform`.
 - 3 Log on to the Platform Management Console for the first time by specifying
 - ◆ User Name: Admin
 - ◆ Password: Admin

For security in a production environment, we strongly recommend that you change the password of the Admin account.
-

Modify user password and other account information

Prerequisites: You must be a cluster administrator to change the password of the Admin account or make any other modifications to the account.

-
- 1 From the command line, run `egosh user modify -u user_account -x password`, specifying the name of the user account you wish to modify followed by its new password.

The password is made up of one to eight alphanumeric or special characters, except greater than (>), less than (<), ampersand (&), or control characters (Ctrl + key).

NOTE: You cannot modify the name of the user account, only the password.

- 2 If desired, modify other user account information by adding any of the following optional commands to `egosh user modify -u user_account`:
 - ◆ `-e email`: Specifies a new email address of the user to whom this account belongs. Specify up to 64 alphanumeric or special characters, except greater than (>), less than (<), ampersand (&), or control characters (Ctrl + key).
 - ◆ `-t telephone`: Specifies the telephone number of the user to whom this account belongs. Specify up to 20 numbers and spaces.
 - ◆ `-d description`: Specifies any additional information about the user account or the user to whom this account belongs. Specify up to 200 alphanumeric or special characters, except control characters (Ctrl + key). Enclose description in quotation marks if there are spaces within it.

```
egosh user modify -u user_account [-x password] [-e email]
[-t telephone] [-d description]
```

Understanding the administrator dashboard

The dashboard is only visible to cluster and consumer administrators. If you are not logged in as a cluster or consumer administrator, you cannot see the dashboard.

Overview

The summary dashboard is your window into your cluster. Use the dashboard to get an overall picture of the health of your cluster and to get early warning for any systemic problems that may be occurring. The dashboard gives you vital, real-time information about the health of your workload and your hosts.

NOTE: Your dashboard is dynamic and can change depending on what components you have integrated. For information about integration, contact Platform Support.

You can customize your tables on the dashboard and elsewhere by clicking **Preferences**.

Your hosts

The health of your hosts is one of the major elements of your cluster that can seriously affect its efficiency. Healthy hosts mean better throughput and a more stable production environment.

What you see

Your cluster view gives you a quick look at the health of your entire cluster at a glance. Use the **Hosts by State** and **Hosts by CPU Utilization** charts in the Cluster Health area to monitor how your hosts are doing and how hard they are working.

Beneath these charts, you can see the health of your master host and your master candidates if you have specified any. For your master host health, rest your mouse over the icon for a detailed description of its real-time attributes or click on the icon for detailed information about that particular host.

What to watch for

For your cluster to be healthy and working efficiently, your hosts have to be running or available to run workload units. They must also be able to run workload units at an acceptable CPU utilization level. Use the Platform Management Console to monitor for

- ◆ A large number of closed or unavailable hosts
- ◆ Hosts running at very high or very low CPU utilization

Either of these indicators means that your cluster is not running as efficiently as possible and may need some attention.

CPU utilization levels: What is right?

Only you can decide what the optimal CPU utilization level is for your workload. In some cases, 90% is acceptable and in others 70% is good. Try to recognize the level at which your hosts are attaining the highest CPU utilization level without becoming frequently unavailable.

Goal 1: Predictability and output

If you want your cluster to be as predictable as possible and for workload units to finish and produce results at a possible cost of speed, make sure your hosts are only running at 70% CPU utilization or less.

Goal 2: Fast turnaround time

If what you require is speed (you do not care if some workload units fail as long as most of them finish as quickly as possible), then 90% CPU utilization is probably the right level for you.

Understanding the host view

The host view in the Platform Management Console gives you all the information you need to keep an eye on your entire cluster and to quickly and easily spot those hosts that are experiencing any difficulties. Use **Filter Settings** to customize the view of your hosts. For example, you can filter hosts by state to display only unavailable or closed hosts.

You can also select **Hosts (List View)** and customize the metrics you see in the table with **Preferences**.

Healthy hosts mean that your workload is running optimally and you are making the best possible use of your resources. If your hosts are closed or unavailable, your workload suffers. The more hosts you have available to run workload units, the more robust your cluster becomes. The first step in keeping your cluster healthy is to monitor the health of your hosts.

For details about the performance of a specific host, rest your cursor on the icon (when using the Icon View). In either the Icon View or the List View, click on the icon or host name to get host properties.

Master host

Your master host is always displayed as an icon at the bottom of your host view. For details, rest your cursor on the icon. For master host properties, click the icon.

How can I tell my hosts are in trouble?

Your cluster only works as well as your hosts are performing. But how do you know when hosts are not working as well as possible? Use the hints below to get an idea of how you can use the Platform Management Console to monitor your host health.

Pattern

Use the icon view to display your hosts and their health visually. It is the pattern of the health of the hosts that is important to maintaining a healthy cluster. There should be few hosts Unavailable or Closed compared to OK hosts.

Height

The color height of the icons in the icon view gives you the CPU utilization of that host. Generally, in a healthy cluster, most hosts have between 7 and 9 bars in height, meaning they are using between 70% and 90% CPU utilization. Depending on the workload you are running, these thresholds may not be correct for you; they are general guidelines only.

Master host

Keep an eye on your master host at the bottom of the Host View. It should not be overloaded. Candidates are designed to back up your cluster should a master host become unavailable for any reason, but it is recommended that you safeguard the health of your master host as much as possible.

Set the command-line environment

On Linux hosts, set the environment before you run any LSF or EGO commands. You need to do this once for each session you open. `root`, `lsfadmin`, and `egoadmin` accounts use LSF and EGO commands to configure and start the cluster.

You need to reset the environment if the environment changes during your session, for example, if you run `egoconfig mghost`, which changes the location of some configuration files.

-
- ◆ For `csh` or `tcsh`, use `cshrc.lsf`.
`source LSF_TOP/conf/cshrc.lsf`
 - ◆ For `sh`, `ksh`, or `bash`, use `profile.lsf`:
`. LSF_TOP/conf/profile.lsf`
-

If Platform EGO is enabled in the LSF cluster (`LSF_ENABLE_EGO=Y` and `LSF_EGO_ENVDIR` are defined in `lsf.conf`), `cshrc.lsf` and `profile.lsf` set the following environment variables.

- ◆ `EGO_BINDIR`
- ◆ `EGO_CONFDIR`
- ◆ `EGO_ESRVDIR`
- ◆ `EGO_LIBDIR`
- ◆ `EGO_LOCAL_CONFDIR`
- ◆ `EGO_SERVERDIR`
- ◆ `EGO_TOP`

See the *Platform EGO Reference* for more information about these variables.

See the *Platform LSF Configuration Reference* for more information about `cshrc.lsf` and `profile.lsf`.

View and modify cluster properties

You must be an administrator to change any cluster properties.

You can view cluster properties including

- ◆ Cluster name
- ◆ User accounts who have access to the cluster as administrators
- ◆ Master candidates
- ◆ Resource groups you have assigned to your cluster

You can view or modify cluster properties. Only the user accounts assigned to be cluster administrators can be modified.

To view cluster properties

-
- 1 Click **Cluster Settings**.
The Summary page displays.
 - 2 Click **Cluster Properties**.
-

To modify cluster properties

You cannot remove the `egoadmin` account as a cluster administrator.

- 1 Follow the instructions for viewing cluster properties.
- 2 Change which user accounts you wish to assign as cluster administrators.
Specify administrators for this cluster by using the **Add** and **Remove** buttons to transfer user accounts.
- 3 Click **Apply**.
You cannot click **Apply** until you have changed the list of administrators.

Build your resource distribution tree

Follow these rules when building a resource distribution tree:

- ◆ You must be a cluster administrator.
- ◆ You cannot create a consumer beneath a consumer that has anything registered to it. You must unregister before you can create a sub-consumer.
- ◆ You can only create five levels of consumers including the cluster (top) level.
- ◆ You cannot change your cluster name.
- ◆ You must build your tree from the top down.

Defaults

The consumers `SampleApplications` and `ManagementServices`, along with their sub-consumers, are installed by default.

- ◆ `SampleApplications` has one sub-consumer, `EclipseSamples`, with several SDK Example leaf consumers beneath it.
The `SampleApplications` consumer and its sub-consumer can be modified or deleted (although you will want to use the provided `EclipseSamples` in order to begin using Platform EGO right away—this is a sample consumer with registered applications that are ready to run).
- ◆ The `ManagementServices` consumers runs essential services and should never be modified or deleted.

Create a consumer

Prerequisites: To create a consumer, you must either be a cluster administrator or a consumer administrator for the branch on which you are creating a consumer.

- 1 Click **Consumers & Plans**.
A list of existing top-level consumers in your tree displays.
- 2 Locate and click on the tree level for which you would like to add a consumer.
As a best practice, synchronize the order of consumers (as reflected in the consumer tree) with the Consumer Rank you may set at a later time. This way resources will be lent and reclaimed according to a common measure of priority.
- 3 From **Global Actions**, select **Create a Consumer**.
The Create a Consumer page displays.

You can create a consumer at any level of your existing tree, except where a consumer already has something registered to it; in this case, no other consumers can be created below it in the tree.

4 Fill in the consumer properties.

Some of these properties may be already filled out or disabled depending on which tree level you are adding a consumer to.

- a Specify a name for your new consumer.
- b (Optional; Available only at the level on your tree just beneath your cluster.) If asked, you can choose one or more administrators for this consumer.

Specified administrators automatically become administrators for any other consumers created on this branch. You cannot add a cluster administrator as a consumer administrator.

- c Specify zero, one, or more users for this consumer.

You must specify at least one user in order for the consumer to run workload units.

- d Specify the workload execution user account (the OS account under which workload units run).

This field may be pre-populated, but you can modify it. Windows accounts should include a domain name.

If you specify a Windows user account that has not already been configured, you will have to log on to Platform EGO as the cluster administrator and then run `egosh ego execpasswd` to register the Windows execution password before the execution user can run an activity without exiting. The password must be 31 characters or less.

- e Specify one or more resource groups this consumer should have access to.

You should specify the ManagementHosts resource group plus at least one other or else workload units will not run for this consumer. Only the resource groups specified by your parent in the tree are available for selection.

By default, ManagementHosts and ComputeHosts resource groups are selected. If you have not modified your resource groups, keep these settings.

- f Specify a reclaim grace period.

The reclaim grace period is applied when a resource belonging to another consumer is now being reclaimed by its owner consumer. Setting the reclaim grace period high (compared to the average length of your workload) allows workload units to finish before the resource is reclaimed. Setting the reclaim grace period to zero kills all workload units running and reclaims the resource immediately.

5 Click **Create**.

Postrequisites: You may need to configure the Windows password of the execution user account.

For more information

See *Administering and Using Platform EGO* for detailed steps for creating consumers and examples to begin building your tree.

Manage services

About services

A service is a self-contained, continuously running process that accepts one or more requests and returns one or more responses. Services may have multiple concurrent service instances running on multiple hosts. All Platform EGO services are automatically enabled by default at installation.

By default, management services that are installed with Platform EGO include the following:

- ◆ Service director (`ServiceDirector`): Service that functions as a locating mechanism for other Platform EGO services.
- ◆ Platform Management Console (`WEBGUI`): Service that runs the Platform Management Console.
- ◆ Web service gateway (`WebServiceGateway`): Service that provides a standards-based web services interface for web service clients (applications) to contact Platform EGO.

By default, Platform management services are registered to consumers in the `ManagementHosts` resource group. You do not need to register non-Platform EGO services to consumers in the `ManagementHosts` resource group, although it is recommended that you do (management hosts are not expected to execute workload units for users, but are configured to run important services instead).

Each started service requires one slot, assuming the service is configured for one instance (default setting). By default, Platform EGO allots five slots to run services, although only three are actually used out-of-the-box by the installed management services listed above. You must retain at least three of the five slots for use by the Platform EGO services; the other two can be used by non-Platform EGO services that you might register requiring a management host.

Slots assigned to running services on consumers in the `ManagementHosts` resource group do not have to be on the master host (for example, any management host in the cluster can be the web server); the cluster determines where the service instance runs at startup.

Registered services do not require a host, only the service instance that it runs.

Service states

All Platform EGO services are in one of the following states:

State	Description
DEFINED	The service definition is loaded or created by API without syntax error. This state persists if: <ul style="list-style-type: none"> ◆ You defined a service to be started manually ◆ You have disabled it by API ◆ You stop the service using <code>egosh</code>
INIT	Service is being initialized. Dependencies are being checked. A transitional state before becoming <code>ALLOCATING</code> .

State	Description
ALLOCATING	Allocating resources to the service. A transitional state before becoming STARTED. (Note: A service could remain in this state for some time if resources are not available to start it.)
STARTED	The service is active and running. Services instances are running (the minimum number have been started).
ERROR	Error has been detected. Needs manual debugging.
DEALLOCATING	The service has been disabled by the system. Clean up is occurring. A transitional state before becoming DEFINED.

Service profile

A service profile is an XML file that contains the service definition for a service; when you update a service, you are updating the service profile containing the service definition. Every service, whether it is installed by default or added by an administrator, has a service profile. The XML file lets you configure properties for the service such as

- ◆ The maximum and minimum number of service instances required by the service
- ◆ A service instance description, which describes how to start a service instance for this service
- ◆ Any resources required to run the service instances for this service
- ◆ Any dependencies this service has (such as another service being started)

TIP: We do not recommend changing the XML file for any of the services installed by default unless specifically told to do so. You may need to change ports.

Service properties

Service property names (including full tooltip names) and descriptions are outlined below.

Property	Description
Name (Service name)	Name of service.
State (Service state)	The state of the service: DEFINED, INIT, ALLOCATING, STARTED, ERROR, DEALLOCATING.
# of Inst (Number of service instances)	Number of service instances currently running.
Min Inst (Minimum number of service instances)	Minimum number of instances for this service; set in the service profile. Once the minimum is reached, the state changes to STARTED.
Max Inst (Maximum number of service instances)	Maximum number of instances that can run concurrently for this service; set in the service profile.
Consumer (Consumer this service is registered to)	The ID for the consumer this service is registered under.
RG (Resource group)	The resource group name specified in the service profile.
Description (Optional description)	An optional description field for this service added in the service profile.

Property	Description
Inst/Slot (Maximum number of instances per slot)	Maximum number of instances per slot; set in the service profile.
Res Req (Resource requirements)	Resource requirements for this service to run; set in the service profile.
Type (Start type)	Start type: automatic or manual. An automatic service starts automatically with your cluster. Manual services need to be started manually and do not restart with the cluster. All EGO services are automatic.
Depend (Dependency on starts)	Any dependencies required for this service to start; set in the service profile.
Restarts (Maximum number of restarts)	Maximum number of restarts; set in the service profile.

About service instances

Service instances are the result of a running service. For example, services that are stopped for any reason do not have service instances.

A Platform LSF service instance only runs one activity. You can't control the activity directly, only the service instance which contains it. For example, you can suspend an instance (and therefore the activity it contains), but you cannot suspend the activity directly.

You can have multiple service instances running under a single service; this means multiple activities can run simultaneously. For example, if Platform LSF is running under other clustered application managers (for example, under Platform Symphony or LSF), there could be multiple activities running in each non-EGO service instance.

By default, services have a configured number of service instances to run on each slot (1 instance per slot).

Service instances require a host to run an activity. A service instance runs on the resource group (for example, the ManagementHost group or the ComputeHost group) that is defined in the service's definition file, under

`LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`, or
`LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`.

Service instance states

All service instances have one of the following states:

State	Description
Start	The service instance is starting.
Run	The service instance is running.
Finish	The service instance has completed its task and has finished.

State	Description
Unknown	The service instance has not communicated and its current status is unknown. The execution host is unavailable for some reason. The activity will be restarted on a new host if HostFailoverInterval is defined and reached.
Suspend	The service instance has been suspended by the system. It will automatically be started again when required.

A service instance state is taken from the state of the activity that is running under it.

Service instance definitions

Service instance configurations are specified at the service level. You can specify

- ◆ the maximum number of instances allowed to run concurrently
- ◆ the maximum number of instances per slot
- ◆ the maximum number of instances per host

TIP: When configuring these three parameters, the maximum number of instances should be a multiple of the maximum number of instances per slot which in turn should be a multiple of maximum number of instances per host.

Service instance properties

Service instance property names (including full tooltip names) and descriptions are outlined below.

Property	Description
Sequence (Service instance sequence)	The identification number for the service instance. Values are not configurable and can be from 1 to the maximum number of instances specified for this service. The number is reassigned if the instance is restarted, or if the service is migrated or enabled.
Name (Service name)	Name of service.
State (Service instance state)	The state of the service instance: START, RUN, FINISH, UNKNOWN, SUSPEND.
Start (Start time)	The time the service instance started.
Host (Running host name)	The host on which the service instance is running.

View a consumer's service

Prerequisites: You must be a cluster administrator.

You can view the services that you (or the system) associated with a particular consumer. You can also view services for all consumers. Services are associated with consumers because they must exist somewhere in the resource distribution tree.

- 1 Click **Service Controller**.
The Services page displays.
- 2 In the resource distribution tree, select the consumer you want.

You can get a list of all services for all consumers by clicking on the cluster name. Or you can view the services of one consumer by clicking on that consumer name. A service can only be registered to a leaf. If your consumer has a sub-consumer, it is not a leaf.

Determine the host address where the service is running

Prerequisites: You need to know the address of where the service director (DNS server) is running (contact IT for assistance, if required).

EGO services may not all run on the same management host. You can use `nslookup` to find the address of the host where a specific service is running.

- 1 Using the CLI, type `nslookup`.
- 2 Type `server` and then enter the IP address of the service director (DNS server). The Default Server and Address will return. For example,


```
> server 172.25.237.37
Default server: 172.25.237.37
Address: 172.25.237.37#53
```
- 3 Enter the name of the service for which you want to find the host address. For example,


```
> WEBGUI.ego
Server:          172.25.237.37
Address:        172.25.237.37#53

Name:   WEBGUI.ego
Address: 172.25.237.37
```

Troubleshoot service error states

If you receive a service error message, or a message indicating that a service will not transition out of the `Allocating` state, there are steps you can perform to troubleshoot the issue.

Responding to service message Error

Normally, Platform EGO attempts to start a service multiple times, up to the maximum threshold set in the service profile XML file (containing the service definition). If the service cannot start, you will receive a service error message.

- 1 Try stopping and then restarting the service.
- 2 Review the appropriate service instance log file to discover the cause of the error.

Platform EGO service log files include those for the service director (`ServiceDirector`), web service gateway (`WebServiceGateway`), and the Platform Management Console (`WEBGUI`). If you have defined your own non-EGO services, you may have other log files you will need to review, depending on the service which is triggering the error.

Responding to service message *Allocating*

Allocating is a transitional service state before the service starts running. If your service remains in this state for some time without transitioning to *Started*, or cycles between *Defining* and *Allocating*, you will want to discover the cause of the delay.

- 1 If you are the cluster administrator, review the allocation policy.
 - a Open the service profile XML file (containing the service definition).
 - b Find the consumer for which the service is expected to run.
 - c Ensure that a proper resource plan is set for that consumer.

During a service's "allocation" period, Platform EGO attempts to find an appropriate resource on which to run the service. If it cannot find the required resource, the service will not start.

Manage hosts

Important host roles

Hosts in the cluster may be described as the master host, master candidates, management hosts, compute hosts, or the web server host.

Master host

A cluster requires a master host. This is the first host installed. The master host controls the rest of the hosts in the grid.

Master candidates

There is only one master host at a time. However, if the master host ever fails, another host automatically takes over the master host role, allowing work to continue. This process is called failover. When the master host recovers, the role switches back again.

Hosts that can act as the master are called master candidates. This includes the original master host and all hosts that can take over the role in a failover scenario. All master candidates must be management hosts.

Master host failover

During master host failover, the system is unavailable for a few minutes while hosts are waiting to be contacted by the new master.

The master candidate list defines which hosts are master candidates. By default, the list includes just one host, the master host, and there is no failover. If you configure additional candidates to enable failover, the master host is first in the list. If the master host becomes unavailable, the next host becomes the master. If that host is also unavailable, the next host is considered to become the master, and so on down the list. A short list with two or three hosts is sufficient for practical purposes.

For failover to work properly, the master candidates must share a file system and the shared directory must always be available.

IMPORTANT: The shared directory should not reside on a master host or any of the master candidates. If the shared directory resides on the master host and the master host fails, the next candidate cannot access the necessary files.

Management host

Management hosts belong to the `ManagementHosts` resource group. These hosts are not expected to execute workload units for users. Management hosts are expected to run services such as the web server and web services gateway. The master host and all master candidates must be management hosts.

A slot is the basic unit of resource allocation, analogous to a "virtual CPU".

Management hosts share configuration files, so a shared file system is needed among all management hosts.

A management host is configured when you run `egoconfig mgghost` on the host. The tag `mg` is assigned to the management host, in order to differentiate it from a compute host.

Compute host

Compute hosts are distributed to cluster consumers to execute workload units. By default, compute hosts belong to the `ComputeHosts` resource group.

The `ComputeHosts` group excludes hosts with the `mg` tag, which is assigned to management hosts when you run `egoconfig mgghost`. If you create your own resource groups to replace `ComputeHosts`, make sure they also exclude hosts with the `mg` tag.

By default, the number of slots on a compute host is equal to the number of CPUs.

Web server host

The web server is the host that runs the Platform Management Console. There is only one web server host; it does not need to be a dedicated host. Any management host in the cluster can be the web server (decided when the cluster starts up).

Set your master candidates

Prerequisites: You must be a cluster administrator.

You can modify the order of the hosts in the candidate list (this is the list which specifies the management host(s) in line to become the new master in situations where the existing master host becomes unavailable). You can also change the master host.

- 1 Click **Cluster Settings**.
The Summary of your cluster settings displays.
- 2 Select **Master Candidates**.
- 3 To add hosts to your candidate list, select one or more hosts and click **Add**.
Use SHIFT or CTRL while you are clicking to select more than host at a time.
- 4 To remove any hosts from the candidate list, click the host name and **Remove**.
You cannot remove the master host from this list. The master host is at the top of the list by default when you navigate to this page.
- 5 Change the order of the candidates using **Up** and **Down**.
You can move the master host from the top of the list, however you cannot remove master host from the list.
- 6 When you have the hosts you want in the order you want, click **Apply**.

Changing and saving the order of your candidates restarts your cluster.

The hosts you want as master and master candidates are now set in the order you want them to failover. Your cluster automatically restarts so the changes take effect.

Manage users

User accounts

A user account is a Platform system user who can be assigned to any role for any consumer in the tree. User accounts include optional contact information, a name, and a password.

Once you have user accounts set up, you can assign a user account to a role for each consumer (such as cluster administrator or consumer user).

RESTRICTION: Only cluster administrators have access to **User Accounts** in the Platform Management Console.

EGO user roles

There are three user roles that can be assigned to any user account. Each user role has a specific level of accessibility and control in the Platform Management Console.

Role	Description
Cluster Administrator	A "super user" able to accomplish all administrative and workload tasks, with access to all areas of the Platform Management Console and all actions within it. Entry to the Platform Management Console is through the dashboard, a heads up display that gives you a running summary of the health of your cluster.
Consumer Administrator	Access and control only over own branch of the tree. Consumer administrators are assigned at the top-level consumer and they are administrators for all sub-consumers in that branch of the tree. Entry to the Platform Management Console is through the dashboard, a heads up display that gives you a running summary of the health of your cluster.
Consumer User	Access and control over their own workload units only. Consumer users are assigned to individual consumers on your tree.

You can assign one user role to different user accounts for different consumers.

Add a consumer administrator

Prerequisites: You must create at least one consumer before you can add a consumer administrator role to it. You can also add a consumer administrator *during* the creation of your first consumer.

Consumer administrators are mid-level administrators that may not have access to the entire cluster but only to specific branches of the resource distribution tree. They administer performance and planning for specific consumers and all consumers in the same branch. A consumer administrator is a user account set as a consumer administrator during the creation or modification of a consumer.

TIP: You can only add or remove a consumer administrator at the top-level consumer. The user account you specify as a consumer administrator is automatically an administrator for all consumers in that branch.

- 1 Create a new user account.
- 2 Go to **Consumer & Plans**. Existing consumers display.
- 3 Select the top-level consumer you want to assign a consumer administrator to. Alternatively, you can create a new consumer from the **Global Actions** list. Depending on your action, either the Consumer Properties or Create a Consumer page displays.

- 4 From the section **Specify administrators for this consumer**, select the user account from the list and click **Add**.

The user account you selected is added to the list of administrators for this consumer.

You can specify as many administrators as you want for a consumer. Move multiple user accounts to the administrator list at once by holding down either Shift or Ctrl while you click.

- 5 Click **Apply** (or **Create**) to save your changes.

Your page closes and the list of consumers and their properties refreshes.

You have added a consumer administrator to a top-level consumer and any current or future consumers in this branch.

Add a consumer user

Prerequisites: You must create at least one consumer before you can add a user to it. You can add a user *during* the creation of your first consumer. To add a consumer user to a consumer, you must be a consumer administrator assigned to this consumer or a cluster administrator.

Consumer users are assigned to all consumers. A consumer user only has access to those consumers they have been assigned to. A consumer user is a user account set as a consumer user during the creation or modification of a consumer.

- 1 Create a new user account.
- 2 Go to **Consumer & Plans**.
Any existing consumers display.
- 3 Select the consumer you want to assign a consumer user to. Alternatively, you can create a new consumer from the **Global Actions** list.

Depending on your action, either the Consumer Properties or Create a Consumer page displays.

- 4 From the section **Specify users for this consumer**, select the user account from the list and click **Add**.

The user account you selected is added to the list of users for this consumer.

- 5 Click **Apply** (or **Create**) to save your changes.

Your page closes and the list of consumers and their properties refreshes.

Logging and troubleshooting

LSF log files

LSF event and account log location

LSF uses directories for temporary work files, log files and transaction files and spooling.

LSF keeps track of all jobs in the system by maintaining a transaction log in the work subtree. The LSF log files are found in the directory `LSB_SHAREDIR/cluster_name/logdir`.

The following files maintain the state of the LSF system:

lsb.events LSF uses the `lsb.events` file to keep track of the state of all jobs. Each job is a transaction from job submission to job completion. LSF system keeps track of everything associated with the job in the `lsb.events` file.

lsb.events.n The events file is automatically trimmed and old job events are stored in `lsb.event.n` files. When `mbatchd` starts, it refers only to the `lsb.events` file, not the `lsb.events.n` files. The `bhist` command can refer to these files.

LSF error log location

If the optional `LSF_LOGDIR` parameter is defined in `lsf.conf`, error messages from LSF servers are logged to files in this directory.

If `LSF_LOGDIR` is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.

If `LSF_LOGDIR` is not defined, errors are logged to the system error logs (`syslog`) using the `LOG_DAEMON` facility. `syslog` messages are highly configurable, and the default configuration varies widely from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`.

If the error log is managed by `syslog`, it is probably already being automatically cleared.

If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to `syslog`. If you cannot find any error messages in the log files, they are likely in the `syslog`.

LSF daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The error log file names for the LSF system daemons are:

- ◆ `res.log.host_name`
- ◆ `sbatchd.log.host_name`
- ◆ `mbatchd.log.host_name`
- ◆ `mbschd.log.host_name`

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons is controlled by the parameter `LSF_LOG_MASK` in `lsf.conf`. Possible

values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical.

LSF log directory permissions and ownership

Ensure that the permissions on the `LSF_LOGDIR` directory to be writable by `root`. The LSF administrator must own `LSF_LOGDIR`.

EGO log files

Log files contain important run-time information about the general health of EGO daemons, workload submissions, and other EGO system events. Log files are an essential troubleshooting tool during production and testing.

The naming convention for most EGO log files is the name of the daemon plus the host name the daemon is running on.

The following table outlines the daemons and their associated log file names. Log files on Windows hosts have a `.txt` extension.

Daemon	Log file name
ESC (EGO Service Controller)	<code>esc.log.hostname</code>
named	<code>named.log.hostname</code>
PEM (Process Execution Manager)	<code>pem.log.hostname</code>
VEMKD (Platform LSF Kernel Daemon)	<code>vemkd.log.hostname</code>
WSM (Platform Management Console/WEBGUI)	<code>wsm.log.hostname</code>
WSG (Web Service Gateway)	<code>wsg.log</code>

Most log entries are informational in nature. It is not uncommon to have a large (and growing) log file and still have a healthy cluster.

EGO log file locations

By default, most Platform LSF log files are found in `LSF_LOGDIR`, with these exceptions:

- ◆ The service controller log files are found in `LSF_CONFDIR/ego/cluster_name/eservice/esc/log` (Linux) or `LSF_CONFDIR\ego\cluster_name\eservice\esc\log` (Windows).
- ◆ Platform Management Console log files (WSM and Catalina) are found in `LSF_TOP/gui/1.2/tomcat/logs` (Linux) or `LSF_TOP\gui\1.2\tomcat\logs` (Windows).
- ◆ Web service gateway log files are found in `LSF_CONFDIR/ego/cluster_name/eservice/wsg/log` (Linux), or `LSF_CONFDIR\ego\cluster_name\eservice\wsg\log` (Windows).
- ◆ The service director log files, logged by BIND, are found in `LSF_CONFDIR/ego/cluster_name/eservice/esd/conf/named/namedb/named.log.hostname` (Linux), or `LSF_CONFDIR\ego\cluster_name\eservice\esd\conf\named\namedb\named.log.hostname` (Windows)

Log files can also be accessed through the Platform Management Console (from within the **Logging** tab).

EGO log entry format

Log file entries follow the format

date time_zone log_level [process_id:thread_id] action:description/message

where the date is expressed in YYYY-MM-DD hh-mm-ss.sss.

For example, 2006-03-14 11:02:44.000 Eastern Standard Time ERROR [2488:1036] vemkdexit: vemkd is halting.

EGO log classes

Every log entry belongs to a log class. You can use log class as a mechanism to filter log entries by area. Log classes in combination with log levels allow you to troubleshoot using log entries that only address, for example, configuration.

Log classes are adjusted at run time using `egosh debug`.

Valid logging classes are as follows:

Class	Description
LC_ALLOC	Logs messages related to the resource allocation engine
LC_AUTH	Logs messages related to users and authentication
LC_CLIENT	Logs messages related to clients
LC_COMM	Logs messages related to communications
LC_CONF	Logs messages related to configuration
LC_CONTAINER	Logs messages related to activities
LC_EVENT	Logs messages related to the event notification service
LC_MEM	Logs messages related to memory allocation
LC_PEM	Logs messages related to the process execution manager (pem)
LC_PERF	Logs messages related to performance
LC_QUERY	Logs messages related to client queries
LC_RECOVER	Logs messages related to recovery and data persistence
LC_RSRC	Logs messages related to resources, including host status changes
LC_SYS	Logs messages related to system calls
LC_TRACE	Logs the steps of the program

EGO log levels

There are nine log levels that allow administrators to control the level of event information that is logged.

When you are troubleshooting, increase the log level to obtain as much detailed information as you can. When you are finished troubleshooting, decrease the log level to prevent the log files from becoming too large.

Valid logging levels are as follows:

Level	Description
LOG_EMERG	Log only those messages in which the system is unusable.
LOG_ALERT	Log only those messages for which action must be taken immediately.
LOG_CRIT	Log only those messages that are critical.

Level	Description
LOG_ERR	Log only those messages that indicate error conditions.
LOG_WARNING	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
LOG_NOTICE	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
LOG_INFO	Log all informational messages and more serious messages.
LOG_DEBUG	Log all debug-level messages.
LOG_TRACE	Log all available messages.

EGO log level and class information retrieved from configuration files

When EGO is enabled, the `pem` and `vemkd` daemons read `ego.conf` to retrieve the following information (as corresponds to the particular daemon):

- ◆ `EGO_LOG_MASK`: The log level used to determine the amount of detail logged.
- ◆ `EGO_DEBUG_PEM`: The log class setting for `pem`.
- ◆ `EGO_DEBUG_VEMKD`: The log class setting for `vemkd`.

The `wsm` daemon reads `wsm.conf` to retrieve the following information:

- ◆ `LOG_LEVEL`: The configured log class controlling the level of event information that is logged (`INFO`, `ERROR`, `WARNING`, or `DEBUG`)

The `wsg` daemon reads `wsg.conf` to retrieve the following information:

- ◆ `WSG_PORT`: The port on which the Web service gateway (WebServiceGateway) should run
- ◆ `WSG_SSL`: Whether the daemon should use Secure Socket Layer (SSL) for communication.
- ◆ `WSG_DEBUG_DETAIL`: The log level used to determine the amount of detail logged for debugging purposes.
- ◆ `WSG_LOGDIR`: The directory location where `wsg.log` files are written.

The service director daemon (`named`) reads `named.conf` to retrieve the following information:

- ◆ `logging, severity`: The configured severity log class controlling the level of event information that is logged (`critical`, `error`, `warning`, `notice`, `info`, `debug`, or `dynamic`). In the case of a log class set to `debug`, a log level is required to determine the amount of detail logged for debugging purposes.

Why do log files grow so quickly?

Every time an EGO system event occurs, a log file entry is added to a log file. Most entries are informational in nature, except when there is an error condition. If your log levels provide entries for all information (for example, if you have set them to `LOG_DEBUG`), the files will grow quickly.

Suggested settings:

- ◆ During regular EGO operation, set your log levels to `LOG_WARNING`. With this setting, critical errors are logged but informational entries are not, keeping the log file size to a minimum.

- ◆ For troubleshooting purposes, set your log level to LOG_DEBUG. Because of the quantity of messages you will receive when subscribed to this log level, change the level back to LOG_WARNING as soon as you are finished troubleshooting.

TIP: If your log files are too long, you can always rename them for archive purposes. New, fresh log files will then be created and will log all new events.

How often should I maintain log files?

The growth rate of the log files is dependent on the log level and the complexity of your cluster. If you have a large cluster, daily log file maintenance may be required. We recommend using a log file rotation utility to do unattended maintenance of your log files. Failure to do timely maintenance could result in a full file system which hinders system performance and operation.

View EGO log files

Event and audit log files can be viewed through the Platform Management Console.

- 1 From the Platform Management Console, navigate to **Logging**.
- 2 Select **Standard Logs**.
- 3 From **Logs (list)**, select a log from the list (for example, "audit").
All logs related to the selected log type are listed for every host in the cluster.
- 4 Under Log retrieval parameters, specify the following:
 - a Control the size of the retrieved file by selecting how much of the log you want to see. Retrieval options include the following:
 - ◆ Complete log
 - ◆ Specified number of lines (if you choose to retrieve a specified number of lines, enter the number in the field beside the drop-down list)
 - ◆ Specified amount (if you choose to retrieve a specified amount, enter the amount, in MB, beside the drop-down list)
 - b Optional. Filter the available hosts by typing the beginning characters of the host name in the Filter available hosts field.
For example, type Win_name00 to list hosts with a host name that includes Win_name00.
The list of available hosts is dynamically filtered as you type to show those hosts containing the same characters/name.
 - c Choose which hosts you which want to retrieve and view this log type from (for example, all hosts for which you want to view the "audit" log from).
 - 1 Click a host name from the Available hosts list.
 - 2 Click **Add - >** to move this host to the Retrieve logs from these hosts list.
 - 3 Do this for as many hosts as you want to retrieve logs from.

(Note that for performance reasons, there is a limit to the number of hosts you can add to the list. A message alerts you when this limit is reached.)

5 Click **Retrieve log**.

A list of logs from the requested hosts displays, along with details including the host on which the log file is located, the log file size, and the date of the last entry in the log file.

6 Click a specific log file to view or save.

Choose to open and view the log file in a specified program, or save it to disk.

NOTE: Log files cannot be edited, renamed, or deleted from within the Platform Management Console.

Troubleshoot using multiple EGO log files

EGO log file locations and content

If a service does not start as expected, open the appropriate service log file and review the run-time information contained within it to discover the problem. Look for relevant entries such as insufficient disk space, lack of memory, or network problems that result in unavailable hosts.

Log file	Default location	What it contains
catalina.out	Linux: LSF_TOP/gui/1.2/tomcat/logs/catalina.out Windows: LSF_TOP\gui\1.2\tomcat\logs\catalina.out	Logs system errors and debug information from Tomcat web server startup.
esc.log	Linux: LSF_CONFDIR/ego/cluster_name/eservice/esc/log/esc.log.hostname Windows: LSF_CONFDIR\ego\cluster_name\eservice\esc\log\esc.log.hostname	Logs service failures and service instance restarts based on availability plans. Errors surrounding Platform Management Console startup are logged here.
named.log	Linux: LSF_CONFDIR/ego/cluster_name/eservice/esd/conf/named/namedb/named.log.hostname Windows: LSF_CONFDIR\ego\cluster_name\eservice\esd\conf\named\namedb\named.log.hostname	Logs information gathered during the updating and querying of service instance location; logged by BIND, a DNS server.
pem.log	Linux: LSF_LOGDIR/pem.log.hostname Windows: LSF_LOGDIR\pem.log.hostname	Logs remote operations (start, stop, control activities, failures). Logs tracked results for resource utilization of all processes associated with the host, and information for accounting or chargeback.

Log file	Default location	What it contains
vemkd.log	Linux: LSF_LOGDIR/vemkd.log. <i>hostname</i> Windows: LSF_LOGDIR\vemkd.log. <i>hostname</i>	Logs aggregated host information about the state of individual resources, status of allocation requests, consumer hierarchy, resources assignment to consumers, and started operating system-level process.
wsg.log	Linux: LSF_CONFDIR/ego/ <i>cluster_name</i> /eservice/wsg/log/wsg.log. <i>hostname</i> Windows: LSF_CONFDIR\ego\ <i>cluster_name</i> \eservice\wsg\log\wsg.log. <i>hostname</i>	Logs service failures surrounding web services interfaces for web service clients (applications).
wsm.log	Linux: LSF_TOP/gui/1.2/tomcat/logs/wsm.log. <i>hostname</i> Windows: LSF_TOP\gui\1.2\tomcat\logs\wsm.log. <i>hostname</i>	Logs information collected by the web server monitor daemon. Failures of the WEBGUI service that runs the Platform Management Console are logged here.

Matching service error messages and corresponding log files

If you receive this message...	This may be the problem...	Review this log file
failed to create vem working directory	Cannot create work directory during startup	vemkd
failed to open lock file	Cannot get lock file during startup	vemkd
failed to open host event file	Cannot recover during startup because cannot open event file	vemkd
lim port is not defined	EGO_LIM_PORT in ego.conf is not defined	lim
master candidate can not set GET_CONF=lim	Wrong parameter defined for master candidate host (for example, EGO_GET_CONF=LIM)	lim
there is no valid host in EGO_MASTER_LIST	No valid host in master list	lim
ls_getmyhostname fails	Cannot get local host name during startup	pem
temp directory (%s) not exist or not accessible, exit	Tmp directory does not exist	pem
incorrect EGO_PEM_PORT value %s, exit	EGO_PEM_PORT is a negative number	pem
chdir(%s) fails	Tmp directory does not exist	esc
cannot initialize the listening TCP port %d	Socket error	esc
cannot log on	Log on to vemkd failed	esc
JAVA_HOME is not defined, exit	WEBGUI service profile is wrong	wsm
failed to get hostname: %s	Host name configuration problem	wsm
event_init () failed	EGO event plugin configuration problem in ego.conf file	wsm
ego.conf_loadeventplug () failed	Event library problem	wsm
cannot write to child	Web server is down or there is no response	wsm
child no reply	Web server is down or there is no response	wsm
vem_register: error in invoking vem_register function	VEM service registration failed	wsg
you are not authorized to unregister a service	Either you are not authorized to unregister a service, or there is no registry client	wsg
request has invalid signature: TSIG service.ego: tsig verify failure (BADTIME)	Resource record updating failed	named

For more information

- ◆ About Platform LSF logging and troubleshooting, see [Error and Event Logging](#) on page 661 and [Troubleshooting and Error Messages](#) on page 671
- ◆ About Platform EGO loggings and troubleshooting, see *Administering and Using Platform EGO*

Frequently asked questions

Question *Does LSF 7 on EGO support upgrade of the master host only?*

Answer Yes

Question *Under EGO Service Controller daemon management mode on Windows, does PEM start sbatchd and res directly or does it ask Windows to start sbatchd and res as Windows Services?*

Answer On Windows, LSF still installs sbatchd and res as Windows services. If EGO Service Controller daemon control is selected during installation, the Windows service will be set up as Manual. PEM will start up the sbatchd and res directly, not as Windows Services.

Question *What's the benefit of LSF daemon management through the EGO Service Controller?*

Answer EGO Service Controller provides High Availability services to sbatchd and res, and faster cluster startup than startup with lsadmin and badmin.

Question *How does the hostsetup script work in LSF 7?*

Answer LSF 7 hostsetup script functions essentially the same as previous versions. It sets up a host to use the LSF cluster and configures LSF daemons to start automatically. In LSF 7, running `hostsetup --top=/path --boot="y"` will check the EGO service definition files sbatchd.xml and res.xml. If res and sbatchd startup is set to "Automatic", the host rc setting will only start lim. If set to "Manual", the host rc setting will start lim, sbatchd, and res as in previous versions.

Question *Is non-shared mixed cluster installation supported, for example, adding UNIX hosts to a Windows cluster, or adding Windows hosts to a UNIX cluster?*

Answer In LSF 7, non-shared installation is supported. For example, to add a UNIX host to a Windows cluster, set up the Windows cluster first, then run `lsfinstall -s -f slave.config`. In `slave.config`, put the Windows hosts in LSF_MASTER_LIST. After startup, the UNIX host will become an LSF host. Adding a Windows host is even simpler. Run the Windows installer, enter the current UNIX master host name. After installation, all daemons will automatically start and the host will join the cluster.

Question *As EGO and LSF share base configuration files, how are other resources handled in EGO in addition to hosts and slots?*

Answer Same as previous releases. LSF 7 mbatchd still communicates with LIM to get available resources. By default, LSF can schedule jobs to make use of all resources started in cluster. If EGO-enabled SLA scheduling is configured, LSF only schedules jobs to use resources on hosts allocated by EGO.

Question *How about compatibility for external scripts and resources like elim, melim, esub and others?*

Answer LSF 7 supports full compatibility for these external executables. `elim.xxx` is started under LSF_SERVERDIR as usual. By default, LIM is located under LSF_SERVERDIR.

Question *Can Platform LSF MultiCluster share one EGO base?*

Answer No, each LSF cluster must run on top of one EGO cluster.

Frequently asked questions

Question *So in Platform LSF MultiCluster, I can only upgrade one cluster to LSF 7?*

Answer Yes

Question *Can EGO consumer policies replace MultiCluster lease mode?*

Answer Conceptually, both define resource borrowing and lending policies. However, current EGO consumer policies can only work with slot resources within one EGO cluster. MultiCluster lease mode supports other load indices and external resources between multiple clusters. If you are using MultiCluster lease mode to share only slot resources between clusters, and you are able to merge those clusters into a single cluster, you should be able to use EGO consumer policy and submit jobs to EGO-enabled SLA scheduling to achieve the same goal.

Cluster Version Management and Patching on UNIX and Linux

IMPORTANT: *For LSF 7 Update 2 only, you cannot use the steps in this chapter to update your cluster from LSF 7 Update 1 to Update 2. You must follow the steps in "Migrating LSF Version 7 to Update 2 on UNIX and Linux" to manually migrate your LSF 7 cluster to Update 2.*

Contents

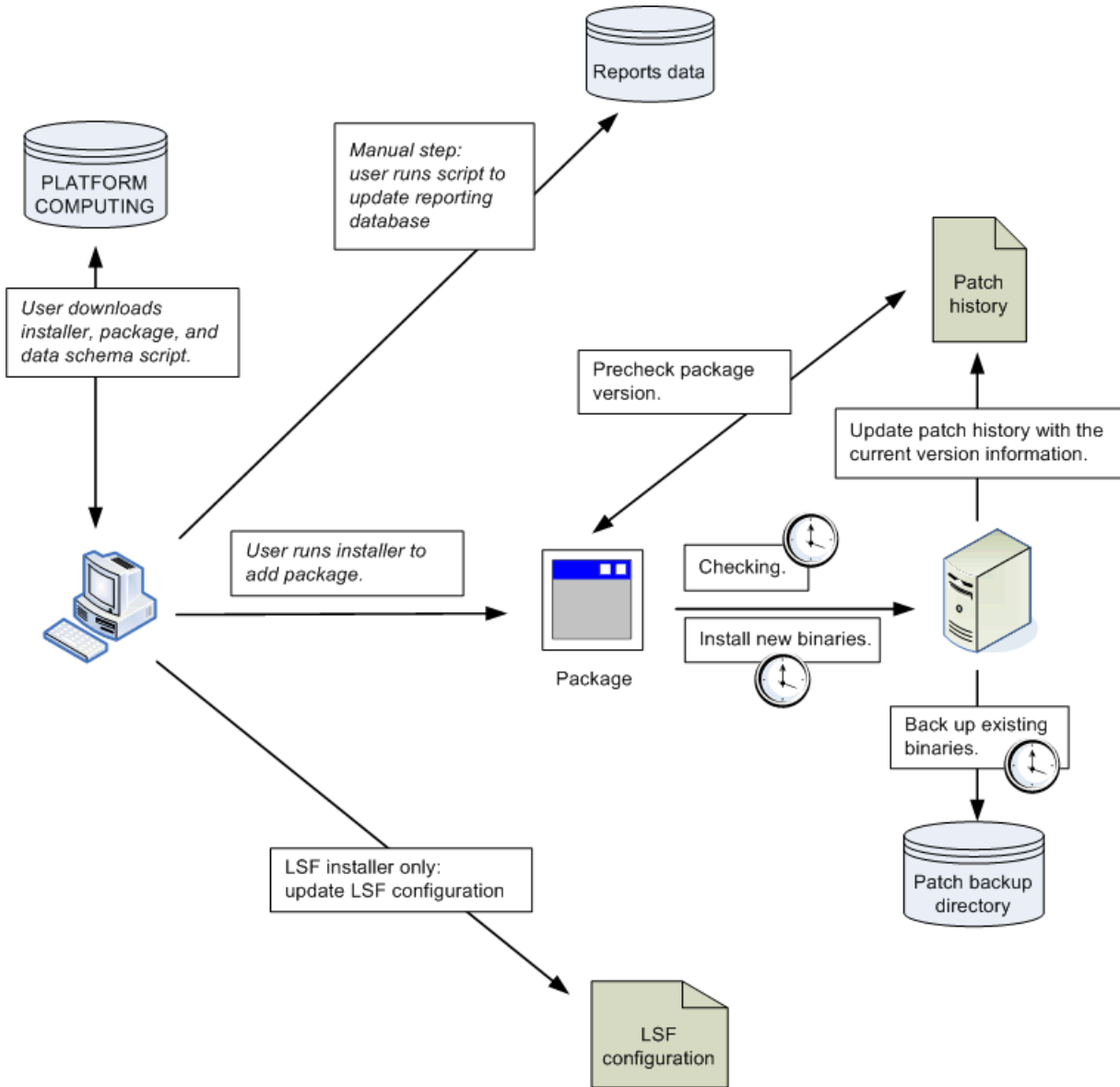
- ◆ [Scope](#) on page 212
- ◆ [Patch installation interaction diagram](#) on page 213
- ◆ [Patch rollback interaction diagram](#) on page 214
- ◆ [Version management components](#) on page 215
- ◆ [Version management concepts](#) on page 217
- ◆ [Cluster patching behavior table](#) on page 219
- ◆ [Cluster rollback behavior table](#) on page 220
- ◆ [Version management files](#) on page 220
- ◆ [Version management commands](#) on page 221
- ◆ [Installing update releases on UNIX and Linux](#) on page 222
- ◆ [Installing fixes on UNIX and Linux](#) on page 223
- ◆ [Rolling back patches on UNIX and Linux](#) on page 223
- ◆ [Patching the Oracle database](#) on page 224
- ◆ [Patching the Derby database](#) on page 225

Scope

Operating system	◆ Supports UNIX hosts within a single cluster
Limitations	pvversions supports LSF Update 1 and later patchinstall supports LSF Update 1 and later For installation of a new cluster, see <i>Installing Platform LSF on UNIX and Linux</i> .

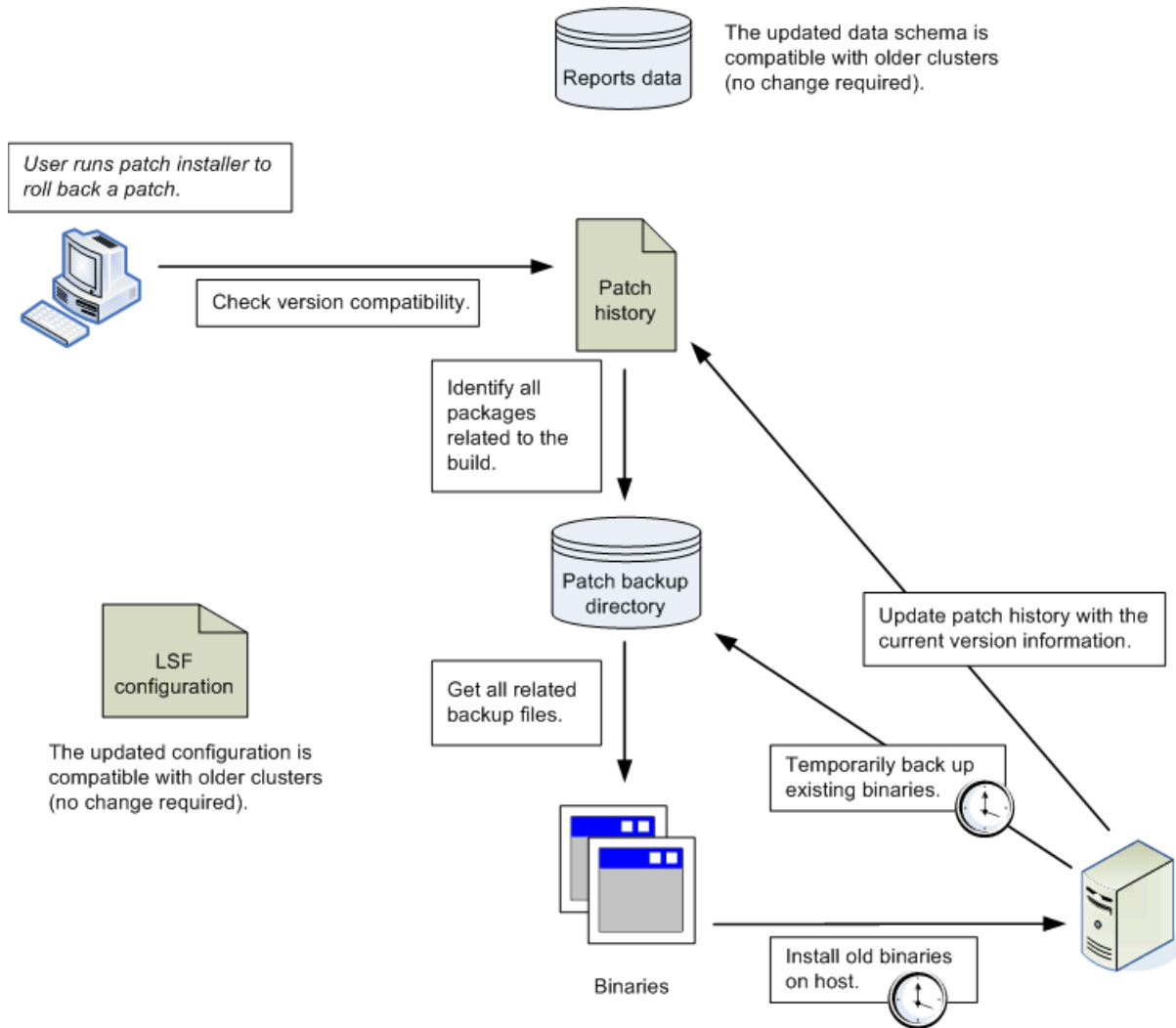
Patch installation interaction diagram

Patches may be installed using the patch installer or LSF installer. The same mechanism is used.



Patch rollback interaction diagram

Use the patch installer to roll back the most recent patch in the cluster.



Version management components

Patches and distributions

Products and versioning

Platform products and components may be separately licensed and versioned. For example, LSF and the Platform Management Console are licensed together, but delivered as separate distributions and patched separately.

Product version is a number identifying the release, such as LSF version 7.0.1. The final digit changes whenever you patch the cluster with a new update release.

In addition to the product version, build date, build number, and binary type are used to identify the distributions. Build number may help identify related distributions for different binary types and is important when rolling back the cluster.

Patching the cluster is optional and clusters with the same product version may have different patches installed, so a complete description of the cluster includes information about the patches installed.

Types of distributions

Upgrades, patches, and hot fixes are used to update the software in an existing cluster.

- ◆ *Product upgrades* deliver a new version of the software with valuable new features. Upgrades require a new license.
- ◆ *Patches* deliver small changes and bug fixes that may result in a minor version change. Patches do not require a new license.
- ◆ *Hot fixes* deliver temporary solutions for emergency problems. Hot fixes do not require a new license.

Types of patches

This document describes installing and removing patches. Patches include fixes, fix packs, and update releases. These do not require a new license.

- ◆ *Update releases*—are full distributions available to all customers at regular intervals and include all fixes intended for general use. Your cluster should always use the latest update release. The same package can be used to patch a cluster or create a new cluster. Each update has a different version number (for example, LSF 7 Update 1 is version 7.0.1).
- ◆ *Fixes*—are partial distributions delivered as needed to resolve customer issues (identified by a specific fix number). Platform Support will advise you if you need install any fixes in your cluster. Installing or removing this type of patch does not change the version of the cluster.
- ◆ *Fix packs (FP)*—contain two or more related fixes in one distribution for your convenience.

Version command

The version command `pversions` is a tool provided to query the patch history and deliver information about cluster and product version and patch levels.

The version command includes functionality to query a cluster or check contents of a package.

The version command is not located with other LSF commands so it may not be in your path. The command location is `LSF_TOP/7.0/install/pversions`

Data schema update script

The update script for your data schema is a tool provided by Platform to update an existing database before you patch the cluster. The update script modifies the data schema so the database is prepared to handle data from an updated cluster.

Use the script that matches your database and cluster version. A patch may not involve any change to the database, or it may require multiple scripts to update different parts of the database.

Patch installer

The patch installer `patchinstall` is a tool provided to install patches on an existing cluster.

The patch installer includes functionality to query a cluster, check contents of a package and compatibility with the cluster, and patch or roll back a cluster.

Patch history

History

The patch history is a record of information about patches installed with the patch installer or the LSF installer, including products and patches installed, dates, and location of backups required for rollback purposes.

The `pversions` command retrieves and displays the version information. The patch installer rollback feature retrieves the backup information.

History directory

The patch history information is kept in the patch history directory. The directory location is `LSF_TOP/patch` by default.

The patch history directory is configurable during installation. See the `PATCH_HISTORY_DIR` parameter in `install.config`.

Patch backups

Backups

The patch installer backs up the current installation before attempting to replace files with the newer versions. The backups are saved so that rollback will be possible later on.

Patches change relatively few files, but for an update release, all the files in the cluster are backed up, so the amount of space required is large. The more patches you install, the more space is required to save multiple backups.

Backup directory

The patch backup files are kept in the patch backup directory. The directory location is `LSF_TOP/patch/backup` by default.

The patch backup directory is configurable during installation. See the `PATCH_BACKUP_DIR` parameter in `install.config`.

Maintenance

Over time, the backups accumulate. You may choose to manually delete old backups, starting with the oldest. Remember that rollback is performed one patch at a time, so your cluster's rollback functionality stops at the point where a backup file is unavailable.

If the backup directory runs out of space, your installations and rollbacks will fail.

You can change your backup directory by setting `PATCH_BACKUP_DIR` in `patch.conf`, but you must copy the contents of the old directory to the new directory manually (or there can be no rollback).

Update release backup control

You can disable backups when installing update releases. In this case, your update is installed without backing up the cluster first, so you cannot remove the update using the rollback functionality.

You might choose this feature to save disk space, to speed up the install process, or if you have your own methods of backing up the cluster.

Backup is always done before installing fixes, so you can always roll back if a fix does not behave as expected.

Multiple daemon files

To make changes without affecting running daemons, the patch installer must move some files to another directory instead of overwriting.

For each file, a new directory is created in parallel with the file. The directory is called `daemons_old`.

Running jobs may require the old files even after you restart the updated cluster.

Version management concepts

Multiple distributions

Like installation, patching the cluster sometimes requires you to download packages for each binary type or product component.

For example, to install an update, you may need to download multiple patches, such as distributions for LSF and distributions for the Platform Management Console.

Depending on the problem, a fix or fix pack may involve changes affecting just one binary type, or multiple distributions to patch multiple binary types.

Order of installation

If you have to install multiple patches, start with the most recent update, which includes all previous fixes. Install on all UNIX hosts to bring the whole cluster up to date. Then install fixes or fix packs as needed.

Installers

The LSF installer installs full distributions and can modify configuration. The LSF installer incorporates the patch installer so the process of updating the files is the same as the patch installer. However, the LSF installer should be used to install an update because the update may require configuration changes that `lsfinstall` can do automatically.

The patch installer installs all patches and never modifies configuration. A partial distribution (FP or fix) can only be installed by the patch installer.

Patch installer accessibility

For clusters version 7.0 or earlier, you must obtain the patch installer separately from Platform, and run the `patchinstall` command from your download directory.

For clusters version 7 Update 1 (7.0.1) or later, the patch installer is available under `install` directory under the LSF installation directory. This location may not be in your path, so run the `patchinstall` command from this directory (`LSF_TOP/7.0/install/patchinstall`).

Version command accessibility

For clusters version 7.0 or earlier, the version command is not available.

For clusters version 7 Update 1 (7.0.1) or later, the command is available under `install` directory under the LSF installation directory (`LSF_TOP/7.0/install/pversions`). It is not located with other LSF commands so it may not be in your path by default.

lsfinstall and install.config versions

The LSF installer may change with each update. You should not install a new update using the old `lsfinstall` program or `install.config` template. To properly update your cluster with new parameters and new configuration, make sure your installers match the version of the distribution you are installing.

Command environment

Both `patchinstall` and `pversions` on UNIX need environment information to identify your cluster.

Before you run the command, set your environment using `profile.lsf` or `cshrc.lsf`. You may have already done this to administer your cluster.

As a workaround, you may use the `-f` option in the command line and specify a file that defines your environment. For more information, see the command reference.

Silent install

The silent install option is used for automated installations.

For `lsfinstall`, enable silent install by the `LSF_QUIET_INST` parameter in `install.config`. Silent install hides some messages.

For `patchinstall`, enable silent install by the `--silent` option in the command line. Silent install shows all messages but does not prompt for confirmations.

Windows-UNIX clusters and Windows clusters

If your cluster has both Windows and UNIX, patch the UNIX hosts in the cluster using the patch installer. Patch the Windows hosts using Windows tools.

The Windows patch files should be installed in order from oldest to newest on every Windows host if you have more than one to install.

To install a Windows patch, double click the .msp file for the OS you want and follow the wizard. You may be asked to reboot after installing. Follow the Windows prompts if applicable.

TIP: You can also install silently.

Cluster patching behavior table

When ...	Actions...	The result ...
Normal behavior.	The installer replaces current files with new.	◆ Success, cluster is updated.
Installing an update and the patch history is missing (files are not found in the directory defined by the parameter PATCH_HISTORY_DIR in patch.conf)	The installer creates new history files in the directory. The installer cannot determine compatibility but installs anyway because an update is a full distribution.	◆ Cluster is modified but if the update is not compatible (a previous version instead of newer version), the cluster may not work properly.
Installing a fix and the patch history is missing (files are not found in the directory defined by the parameter PATCH_HISTORY_DIR in patch.conf)	For a fix, the installer cannot determine compatibility.	◆ No update, cluster remains in same state ◆ Error presented on screen and logged in patch.log and patch.err
The installer is partway through the installation when there is a problem. The cluster contains some older files and some newer files.	If the installer cannot complete, it reverses the update actions, removing the newer files and returning the older ones.	◆ No update, cluster remains in same state. ◆ Error presented on screen and logged
Installing a fix and a file in the cluster is newer than the file in the patch (build number in cluster is larger than build number of patch).	Prompt user to overwrite or preserve file. Install other files in the patch as usual.	◆ Each build of a file is backwards compatible, so this patch works properly with the newer file. ◆ Overwriting the newer file may break functionality of a newer patch in the cluster.
Installing a fix and a file in the cluster has been modified since the last patch (current file size does not match size recorded in patch history).	Prompt user to overwrite or exit.	◆ Overwriting a corrupt file will result in correct behavior. ◆ Overwriting a customized file will break existing functionality. You can modify the updated file manually after installation. ◆ Patch functionality depends on updated content in the new file, so you cannot install the patch if you do not overwrite the file.

Cluster rollback behavior table

When ...	Actions...	The result ...
Normal behavior.	The installer replaces current files with previous backup.	<ul style="list-style-type: none"> ◆ Success, cluster reverts to previous state.
The patch history is missing (files are not found in the directory defined by the parameter <code>PATCH_HISTORY_DIR</code> in <code>patch.conf</code>)	Without the history, the installer cannot determine which backups to use. Since there is nothing to replace them with, the installer does not remove the current files.	<ul style="list-style-type: none"> ◆ No rollback, cluster remains in same state. ◆ Error presented on screen and logged
You did not specify the most recent patch.	The history indicates that the patch is not the newest backup. The installer must use the most recent backup to roll back.	<ul style="list-style-type: none"> ◆ No rollback, cluster remains in same state. ◆ Error presented on screen and logged
The backups are missing (expected files are not found in the directory defined by the parameter <code>PATCH_BACKUP_DIR</code> in <code>patch.conf</code>).	Since there is nothing to replace them with, the installer does not remove the current files.	<ul style="list-style-type: none"> ◆ No rollback, cluster remains in same state. ◆ Error presented on screen and logged
The installer is partway through the roll back when there is a problem. The cluster contains some older files and some newer files.	If the installer cannot complete, it reverses the rollback actions, removing the older files and returning the newer ones.	<ul style="list-style-type: none"> ◆ No rollback, cluster remains in same state. ◆ Error presented on screen and logged

Version management files

Logs

File	Description
<code>patch.log</code>	<p>This file:</p> <ul style="list-style-type: none"> ◆ Created by the patch installer (not created if you use <code>lsfinstall</code>) ◆ Created when you install a patch or update release ◆ Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>) ◆ Logs installation steps
<code>precheck.log</code>	<p>This file:</p> <ul style="list-style-type: none"> ◆ Created by the patch installer ◆ Created when you install or check a patch ◆ Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>) ◆ Logs precheck steps
<code>install.log</code>	<p>This file:</p> <ul style="list-style-type: none"> ◆ Created by the LSF installer (not created if you use <code>patchinstall</code>) ◆ Created when you install a new cluster or update release ◆ Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>) ◆ Logs installation steps

Version management commands

Commands to modify cluster

Command	Description
<code>lsfinstall</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Creates a new cluster (using any full distribution including update releases) ◆ Patches a cluster with an update release (a full distribution) by installing binaries and updating configuration
<code>patchinstall</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Patches a cluster by installing binaries from a full or partial distribution (does not update configuration, so <code>lsfinstall</code> is recommended for an update release)
<code>patchinstall -r</code>	<p>This command</p> <ul style="list-style-type: none"> ◆ Rolls back a cluster by removing binaries (does not roll back configuration, so rollback of updates may not be recommended)

Commands to monitor cluster

Command	Description
<code>pversions</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Displays product version information for the entire cluster, including patch levels ◆ Displays detailed information for specific builds or files in the cluster; for example, see what files were modified after installing a patch
<code>file_name -V</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Displays detailed information for a specific file in the cluster (specify the installed file, for example <code>lim -V</code>)

Commands to check uninstalled packages

Command	Description
<code>pversions -c</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Displays detailed information about the contents of an uninstalled package
<code>patchinstall -c</code>	<p>This command:</p> <ul style="list-style-type: none"> ◆ Tests if an uninstalled package is compatible with the cluster

Installing update releases on UNIX and Linux

To install an update release to the cluster.

IMPORTANT: For LSF 7 Update 2 only, you cannot use the steps in this section to update your cluster from LSF 7 Update 1 to Update 2. You *must* follow the steps in “*Migrating LSF Version 7 to Update 2 on UNIX and Linux*” to manually migrate your LSF 7 cluster to Update 2.

- 1 If you need to patch the reporting database, download the corresponding database update scripts and update the database schema first.
- 2 Download and extract the new version of `lsfinstall`.
For example,

```
zcat lsf7.0.1_lsfinstall.tar.Z | tar xvf -
```
- 3 Prepare the `install.config` file using the new template and information from your original installation. The new template may have new parameters for you to set.
- 4 Download the patches. If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster. Put the distribution files in the same directory as `lsfinstall`.
- 5 Run the new LSF installer.
For example,

```
lsfinstall -f install.config
```


Specify the patches to install and let the installer finish.
- 6 Restart the cluster.
This will make changes to daemons take effect.
- 7 Optional. Run `pversions` to determine the state of the cluster.
- 8 Optional. Free some space by deleting the contents of backup directories under EGO and LSF installation directories.

Installing fixes on UNIX and Linux

To install fixes or fix packs to update the cluster.

- 1 To patch the reporting database, download the corresponding database update scripts and update the database schema first.
- 2 Download the patches from Platform. If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster.

Put the distribution files on any host.

For example,

```
//HostB/downloads/pkg1
```

```
//HostB/downloads/pkg2
```

- 3 Log on to a host in the cluster.
- 4 Set your environment (if you cannot do this, prepare a configuration file and use the `-f` option in the `pversions` and `patchinstall` commands).

```
source LSF_TOP/conf/cshrc.lsf (for csh or tcsh)
```

```
. LSF_TOP/conf/profile.lsf (for sh, ksh, or bash)
```

- 5 Run the patch installer tool and specify the patches to install.

For example,

```
LSF_TOP/7.0/install/patchinstall //HostB/downloads/pkg1
```

```
//HostB/downloads/pkg2
```

Let the patch installer finish.

- 6 If you were prompted to do so, restart the cluster.
Patches that affect running daemons require you to restart manually.
- 7 Optional. Run `LSF_TOP/7.0/install/pversions` to determine the state of the cluster.
- 8 Optional. If you were prompted to restart the cluster and have done so, you can free some space by deleting the contents of `backup` directories under `EGO` and `LSF` installation directories.

Rolling back patches on UNIX and Linux

To remove patches that you installed using `patchinstall`, and return the cluster to a previous state.

- 1 Log on to a host in the cluster.
- 2 Set your environment (if you cannot, prepare a configuration file and use `-f` option in `pversions` and `patchinstall` commands).

```
source LSF_TOP/conf/cshrc.lsf (for csh or tcsh)
```

```
. LSF_TOP/conf/profile.lsf (for sh, ksh, or bash)
```

- 3 Run `LSF_TOP/7.0/install/pversions` to determine the state of the cluster and find the build number of the last patch installed (roll back one patch at a time).
- 4 Run `patchinstall` with `-r` and specify the build number of the last patch installed (the patch to be removed).

```
patchinstall -r 12345
```
- 5 If you were prompted to do so, restart the cluster.
Patches that affect running daemons require you to restart manually.
- 6 If necessary, modify LSF cluster configuration manually. This may be necessary to roll back an update.
- 7 Optional. Run `LSF_TOP/7.0/install/pversions` to determine the state of the cluster.

To roll back multiple builds, repeat as required until the cluster is in the state you want. The database schema is backwards compatible, so you do not need to change the reporting database.

Patching the Oracle database

Prerequisites: The Oracle database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver (`ojdbc14.jar` or newer) for the Oracle database. This driver is available from the following URL:
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html
- ◆ You are able to run `sqlplus`.

To patch the reporting database as part of patching the cluster, get the corresponding database update scripts and update the database schema first.

- 1 When you download the patches for your cluster, download the corresponding database update scripts from Platform.
- 2 In the command console, open the database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/Oracle
```
- 3 Run the scripts to create a database schema.

```
sqlplus user_name/password@connect_string @update_script
```

where
 - ◆ *user_name* is the user name on the database server
 - ◆ *password* is the password for this user name on the database server
 - ◆ *connect_string* is the named SQLNet connection for this database
 - ◆ *update_script* is the name of the patch script

Patching the Derby database

Prerequisites: The Derby database is properly configured and running:

To patch the reporting database as part of patching the cluster, get the corresponding database update scripts and update the database schema first.

1 When you download the patches for your cluster, download the corresponding database update scripts from Platform.

2 In the command console, open the database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/Derby
```

3 Start the `ij` tool.

```
ij.sh
```

4 Connect to the database.

```
connect
```

```
'jdbc:derby://host_name:port/db_name;user=user_name;password=password'
```

where

- ◆ *host_name* is the Derby database host
- ◆ *port* is the Derby database port, 1527 by default
- ◆ *db_name* is the database name, `app` by default
- ◆ *user_name* is the database user name, `app` by default
- ◆ *password* is the database login password, `app` by default

5 Run the scripts to create a database schema.

```
run 'update_script'
```

where

- ◆ *update_script* is the full path to the patch script

Patching the Derby database

II

Working with Resources

- ◆ [Understanding Resources](#) on page 229
- ◆ [Adding Resources](#) on page 249
- ◆ [Managing Software Licenses with LSF](#) on page 261

Understanding Resources

Contents

- ◆ [About LSF Resources](#) on page 230
- ◆ [How Resources are Classified](#) on page 232
- ◆ [How LSF Uses Resources](#) on page 235
- ◆ [Load Indices](#) on page 237
- ◆ [Static Resources](#) on page 240
- ◆ [Automatic Detection of Hardware Reconfiguration](#) on page 246

About LSF Resources

The LSF system uses built-in and configured resources to track job resource requirements and schedule jobs according to the resources available on individual hosts.

View available resources

View cluster resources (lsinfo)

- 1 Use `lsinfo` to list the resources available in your cluster.

The `lsinfo` command lists all the resource names and their descriptions.

```
lsinfo
RESOURCE_NAME  TYPE      ORDER  DESCRIPTION
r15s           Numeric  Inc    15-second CPU run queue length
r1m            Numeric  Inc    1-minute CPU run queue length (alias:cpu)
r15m           Numeric  Inc    15-minute CPU run queue length
ut             Numeric  Inc    1-minute CPU utilization (0.0 to 1.0)
pg             Numeric  Inc    Paging rate (pages/second)
io             Numeric  Inc    Disk IO rate (Kbytes/second)
ls             Numeric  Inc    Number of login sessions (alias: login)
it             Numeric  Dec    Idle time (minutes) (alias: idle)
tmp            Numeric  Dec    Disk space in /tmp (Mbytes)
swp            Numeric  Dec    Available swap space (Mbytes) (alias:swap)
mem            Numeric  Dec    Available memory (Mbytes)
ncpus          Numeric  Dec    Number of CPUs
ndisks         Numeric  Dec    Number of local disks
maxmem         Numeric  Dec    Maximum memory (Mbytes)
maxswp         Numeric  Dec    Maximum swap space (Mbytes)
maxtmp         Numeric  Dec    Maximum /tmp space (Mbytes)
cpuf           Numeric  Dec    CPU factor
rexpri         Numeric  N/A    Remote execution priority
server         Boolean  N/A    LSF server host
irix           Boolean  N/A    IRIX UNIX
hpux           Boolean  N/A    HP_UX
solaris        Boolean  N/A    Sun Solaris
cserver        Boolean  N/A    Compute server
fserver        Boolean  N/A    File server
aix            Boolean  N/A    AIX UNIX
type           String   N/A    Host type
model          String   N/A    Host model
status         String   N/A    Host status
hname          String   N/A    Host name

TYPE_NAME
HPPA
SGI6
SUNSOL
RS6K
NTX86

MODEL_NAME  CPU_FACTOR
DEC3000     10.00
R10K        14.00
PENT200     6.00
IBM350      7.00
```

```
SunSparc    6.00
HP735      9.00
HP715      5.00
```

View host resources (lshosts)

- 1 Run `lshosts` to get a list of the resources defined on a specific host:

```
lshosts hostA
HOST_NAME    type    model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA        SOL732  Ultra2 20.2   2      256M    679M    Yes ( )
```

View host load by resource

- 1 Run `lshosts -s` to view host load by shared resource:

```
lshosts -s
RESOURCE    VALUE    LOCATION
tot_lic     5        host1 host2
tot_scratch 500      host1 host2
```

The above output indicates that 5 licenses are available, and that the shared scratch directory currently contains 500 MB of space.

The VALUE field indicates the amount of that resource. The LOCATION column shows the hosts which share this resource. The `lshosts -s` command displays static shared resources. The `lslload -s` command displays dynamic shared resources.

How Resources are Classified

Resource categories

By values

Boolean resources	Resources that denote the availability of specific features
Numerical resources	Resources that take numerical values, such as all the load indices, number of processors on a host, or host CPU factor
String resources	Resources that take string values, such as host type, host model, host status

By the way values change

Dynamic Resources	Resources that change their values dynamically: host status and all the load indices.
Static Resources	Resources that do not change their values: all resources except for load indices or host status.

By definitions

Site-Defined Resources	Custom resources defined by user sites: external load indices and resources defined in the <code>lsf.shared</code> file (shared resources).
Built-In Resources	Resources that are always defined in LSF, such as load indices, number of CPUs, or total swap space.

By scope

Host-Based Resources	Resources that are not shared among hosts, but are tied to individual hosts, such as swap space, CPU, or memory. An application must run on a particular host to access the resources. Using up memory on one host does not affect the available memory on another host.
Shared Resources	Resources that are not associated with individual hosts in the same way, but are owned by the entire cluster, or a subset of hosts within the cluster, such as floating licenses or shared file systems. An application can access such a resource from any host which is configured to share it, but doing so affects its value as seen by other hosts.

Boolean resources

Boolean resources (for example, `server` to denote LSF server hosts) have a value of one (1) if they are defined for a host, and zero (0) if they are not defined for the host. Use Boolean resources to configure host attributes to be used in selecting hosts to run jobs. For example:

- ◆ Machines may have different types and versions of operating systems.
- ◆ Machines may play different roles in the system, such as file server or compute server.
- ◆ Some machines may have special-purpose devices needed by some applications.
- ◆ Certain software packages or licenses may be available only on some of the machines.

Specify a Boolean resource in a resource requirement selection string of a job to select only hosts that can run the job.

Some examples of Boolean resources:

Resource Name	Describes	Meaning of Example Name
cs	Role in cluster	Compute server
fs	Role in cluster	File server
solaris	Operating system	Solaris operating system
frame	Available software	FrameMaker license

Shared resources

Shared resources are configured resources that are not tied to a specific host, but are associated with the entire cluster, or a specific subset of hosts within the cluster. For example:

- ◆ Floating licenses for software packages
- ◆ Disk space on a file server which is mounted by several machines
- ◆ The physical network connecting the hosts

LSF does not contain any built-in shared resources. All shared resources must be configured by the LSF administrator. A shared resource may be configured to be dynamic or static. In the above example, the total space on the shared disk may be static while the amount of space currently free is dynamic. A site may also configure the shared resource to report numeric, string or Boolean values.

An application may use a shared resource by running on any host from which that resource is accessible. For example, in a cluster in which each host has a local disk but can also access a disk on a file server, the disk on the file server is a shared resource, and the local disk is a host-based resource. In contrast to host-based resources such as memory or swap space, using a shared resource from one machine affects the availability of that resource as seen by other machines. There will be one value for the entire cluster which measures the utilization of the shared resource, but each host-based resource is measured separately.

The following restrictions apply to the use of shared resources in LSF products.

- ◆ A shared resource cannot be used as a load threshold in the `Hosts` section of the `lsf.cluster.cluster_name` file.
- ◆ A shared resource cannot be used in the `loadSched/loadStop` thresholds, or in the `STOP_COND` or `RESUME_COND` parameters in the queue definition in the `lsb.queues` file.

View shared resources for hosts

-
- 1 Run `bhosts -s` to view shared resources for hosts. For example:

```
bhosts -s
RESOURCE      TOTAL    RESERVED  LOCATION
tot_lic        5         0.0      hostA hostB
tot_scratch    00         0.0      hostA hostB
avail_lic      2         3.0      hostA hostB
avail_scratch  100        400.0    hostA hostB
```

The **TOTAL** column displays the value of the resource. For dynamic resources, the **RESERVED** column displays the amount that has been reserved by running jobs.

How LSF Uses Resources

Jobs submitted through the LSF system will have the resources they use monitored while they are running. This information is used to enforce resource usage limits and load thresholds as well as for fairshare scheduling.

LSF collects information such as:

- ◆ Total CPU time consumed by all processes in the job
- ◆ Total resident memory usage in KB of all currently running processes in a job
- ◆ Total virtual memory usage in KB of all currently running processes in a job
- ◆ Currently active process group ID in a job
- ◆ Currently active processes in a job

On UNIX, job-level resource usage is collected through a special process called PIM (Process Information Manager). PIM is managed internally by LSF.

Viewing job resource usage

The `-l` option of the `bjobs` command displays the current resource usage of the job. The usage information is sampled by PIM every 30 seconds and collected by `sbatchd` at a maximum frequency of every `SBD_SLEEP_TIME` (configured in the `lsb.params` file) and sent to `mbatchd`. The update is done only if the value for the CPU time, resident memory usage, or virtual memory usage has changed by more than 10 percent from the previous update, or if a new process or process group has been created.

View load on a host

- 1 Run `bhosts -l` to check the load levels on the host, and adjust the suspending conditions of the host or queue if necessary.

The `bhosts -l` command gives the most recent load values used for the scheduling of jobs. A dash (-) in the output indicates that the particular threshold is not defined.

```
bhosts -l hostB
HOST: hostB
STATUS          CPUF  JL/U  MAX NJOBS  RUN  SSUSP  USUSP  RSV
ok              20.00  2    2    0    0    0    0    0

CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut   pg   io   ls   t   tmp  swp
mem
Total    0.3   0.8  0.9   61%  3.8  72  26  0   6M  253
M 297M
Reserved 0.0   0.0  0.0   0%   0.0  0   0   0   0M  0M
OM

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut   pg   io   ls   it  tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          cpuspeed  bandwidth
```

How LSF Uses Resources

loadSched	-	-
loadStop	-	-

Load Indices

Load indices are built-in resources that measure the availability of dynamic, non-shared resources on hosts in the LSF cluster.

Load indices built into the LIM are updated at fixed time intervals.

External load indices are defined and configured by the LSF administrator. An External Load Information Manager (ELIM) program collects the values of site-defined external load indices and updates LIM when new values are received.

Load indices collected by LIM

Index	Measures	Units	Direction	Averaged over	Update Interval
status	host status	string			15 seconds
r15s	run queue length	processes	increasing	15 seconds	15 seconds
r1m	run queue length	processes	increasing	1 minute	15 seconds
r15m	run queue length	processes	increasing	15 minutes	15 seconds
ut	CPU utilization	percent	increasing	1 minute	15 seconds
pg	paging activity	pages in + pages out per second	increasing	1 minute	15 seconds
ls	logins	users	increasing	N/A	30 seconds
it	idle time	minutes	decreasing	N/A	30 seconds
swp	available swap space	MB	decreasing	N/A	15 seconds
mem	available memory	MB	decreasing	N/A	15 seconds
tmp	available space in temporary file system	MB	decreasing	N/A	120 seconds
io	disk I/O (shown by lsload -l)	KB per second	increasing	1 minute	15 seconds
<i>name</i>	external load index configured by LSF administrator				site-defined

Status

The `status` index is a string indicating the current status of the host. This status applies to the LIM and RES.

The possible values for `status` are:

Status	Description
ok	The host is available to accept remote jobs. The LIM can select the host for remote execution.
-ok	When the status of a host is preceded by a dash (-), it means LIM is available but RES is not running on that host or is not responding.
busy	The host is overloaded (busy) because a load index exceeded a configured threshold. An asterisk (*) marks the offending index. LIM will not select the host for interactive jobs.
lockW	The host is locked by its run window. Use <code>lshosts</code> to display run windows.
lockU	The host is locked by an LSF administrator or root.
unavail	The host is down or the LIM on the host is not running or is not responding.
unlicensed	The host does not have a valid license.

CPU run queue lengths (r15s, r1m, r15m)

The `r15s`, `r1m` and `r15m` load indices are the 15-second, 1-minute and 15-minute average CPU run queue lengths. This is the average number of processes ready to use the CPU during the given interval.

On UNIX, run queue length indices are not necessarily the same as the load averages printed by the `uptime(1)` command; `uptime` load averages on some platforms also include processes that are in short-term wait states (such as paging or disk I/O).

Effective run queue length

On multiprocessor systems, more than one process can execute at a time. LSF scales the run queue value on multiprocessor systems to make the CPU load of uniprocessors and multiprocessors comparable. The scaled value is called the effective run queue length.

Use `lsload -E` to view the effective run queue length.

Normalized run queue length

LSF also adjusts the CPU run queue based on the relative speeds of the processors (the CPU factor). The normalized run queue length is adjusted for both number of processors and CPU speed. The host with the lowest normalized run queue length will run a CPU-intensive job the fastest.

Use `lsload -N` to view the normalized CPU run queue lengths.

CPU utilization (ut)

The `ut` index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no process running has a `ut` value of 0 percent; a host on which the CPU is completely loaded has a `ut` of 100 percent.

Paging rate (pg)

The `pg` index gives the virtual memory paging rate in pages per second. This index is closely tied to the amount of available RAM memory and the total size of the processes running on a host; if there is not enough RAM to satisfy all processes, the paging rate will be high. Paging rate is a good measure of how a machine will respond to interactive use; a machine that is paging heavily feels very slow.

Login sessions (ls)

The `ls` index gives the number of users logged in. Each user is counted once, no matter how many times they have logged into the host.

Interactive idle time (it)

On UNIX, the `it` index is the interactive idle time of the host, in minutes. Idle time is measured from the last input or output on a directly attached terminal or a network pseudo-terminal supporting a login session. This does not include activity directly through the X server such as CAD applications or `emacs` windows, except on Solaris and HP-UX systems.

On Windows, the `it` index is based on the time a screen saver has been active on a particular host.

Temporary directories (tmp)

The `tmp` index is the space available in MB on the file system that contains the temporary directory:

- ◆ `/tmp` on UNIX
- ◆ `C:\temp` on Windows

Swap space (swp)

The `swp` index gives the currently available virtual memory (swap space) in MB. This represents the largest process that can be started on the host.

Memory (mem)

The `mem` index is an estimate of the real memory currently available to user processes. This represents the approximate size of the largest process that could be started on a host without causing the host to start paging.

LIM reports the amount of free memory available. LSF calculates free memory as a sum of physical free memory, cached memory, buffered memory and an adjustment value. The command `vmstat` also reports free memory but displays these values separately. There may be a difference between the free memory reported by LIM and the free memory reported by `vmstat` because of virtual memory behavior variations among operating systems. You can write an ELIM that overrides the free memory values returned by LIM.

I/O rate (io)

The `io` index measures I/O throughput to disks attached directly to this host, in KB per second. It does not include I/O to disks that are mounted from other hosts.

Viewing information about load indices

lsinfo -l

The `lsinfo -l` command displays all information available about load indices in the system. You can also specify load indices on the command line to display information about selected indices:

```
lsinfo -l swp
RESOURCE_NAME: swp
DESCRIPTION: Available swap space (Mbytes) (alias: swap)
TYPE          ORDER  INTERVAL  BUILTIN  DYNAMIC  RELEASE
Numeric      Dec    60        Yes      Yes      NO
```

lsload -l

The `lsload -l` command displays the values of all load indices. External load indices are configured by your LSF administrator:

```
lsload
HOST_NAME  status  r15s  r1m  r15m  ut    pg    ls    it    tmp    swp    mem
hostN      ok      0.0   0.0  0.1   1%   0.0   1     224  43M   67M   3M
hostK      -ok     0.0   0.0  0.0   3%   0.0   3     0    38M   40M   7M
hostF      busy    0.1   0.1  0.3   7%   *17   6     0    9M    23M   28M
hostG      busy    *6.2  6.9  9.5   85%  1.1   30    0    5M    400M  385M
hostV      unavail
```

Static Resources

Static resources are built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time, or when LSF detects hardware configuration changes.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

The resources `ncpus`, `maxmem`, `maxswp`, and `maxtmp` are not static on UNIX hosts that support dynamic hardware reconfiguration.

Static resources reported by LIM

Index	Measures	Units	Determined by
<code>type</code>	host type	string	configuration
<code>model</code>	host model	string	configuration
<code>hname</code>	host name	string	configuration
<code>cpuf</code>	CPU factor	relative	configuration
<code>server</code>	host can run remote jobs	Boolean	configuration
<code>rexpri</code>	execution priority	<code>nice(2)</code> argument	configuration
<code>ncpus</code>	number of processors	processors	LIM
<code>ndisks</code>	number of local disks	disks	LIM
<code>maxmem</code>	maximum RAM	MB	LIM
<code>maxswp</code>	maximum swap space	MB	LIM
<code>maxtmp</code>	maximum space in <code>/tmp</code>	MB	LIM

Host type (`type`)

Host type is a combination of operating system and CPU architecture. All computers that run the same operating system on the same computer architecture are of the same type. You can add custom host types in the `HostType` section of `lsf.shared`. This alphanumeric value can be up to 39 characters long.

An example of host type is `LINUX86`.

Host model (`model`)

Host model is the combination of host type and CPU speed (CPU factor) of your machine. All hosts of the same relative type and speed are assigned the same host model. You can add custom host models in the `HostModel` section of `lsf.shared`. This alphanumeric value can be up to 39 characters long.

An example of host model is `Intel_IA64`.

Host name (`hname`)

Host name specifies the name with which the host identifies itself.

CPU factor (`cpuf`)

The CPU factor (frequently shortened to `cpuf`) represents the speed of the host CPU relative to other hosts in the cluster. For example, if one processor is twice the speed of another, its CPU factor should be twice as large. For multiprocessor hosts,

the CPU factor is the speed of a single processor; LSF automatically scales the host CPU load to account for additional processors. The CPU factors are detected automatically or defined by the administrator.

Server

The `server` static resource is Boolean. It has the following values:

- ◆ 1 if the host is configured to run jobs from other hosts
- ◆ 0 if the host is an LSF client for submitting jobs to other hosts

Number of CPUs (`ncpus`)

By default, the number of CPUs represents the number of physical processors a machine has. As most CPUs consist of multiple cores, threads, and processors, `ncpus` can be defined by the cluster administrator (either globally or per-host) to consider one of the following:

- ◆ Processors
- ◆ Processors and cores
- ◆ Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for `ncpus` is to consider only the number of physical processors (`EGO_DEFINE_NCPUS=procs`).

NOTE: On a machine running AIX, `ncpus` detection is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).

Number of disks (`ndisks`)

The number of disks specifies the number of disks a machine has.

Maximum memory (`maxmem`)

Maximum memory is the total available memory of a machine, measured in megabytes (MB).

Maximum swap (`maxswp`)

Maximum swap is the total available swap space a machine has, measured in megabytes (MB).

Maximum temporary space (`maxtmp`)

Maximum temporary space is the total temporary space a machine has, measured in megabytes (MB).

How LIM detects cores, threads and processors

Traditionally, the value of `ncpus` has been equal to the number of physical CPUs. However, many CPUs consist of multiple cores and threads, so the traditional 1:1 mapping is no longer useful. A more useful approach is to set `ncpus` to equal one of the following:

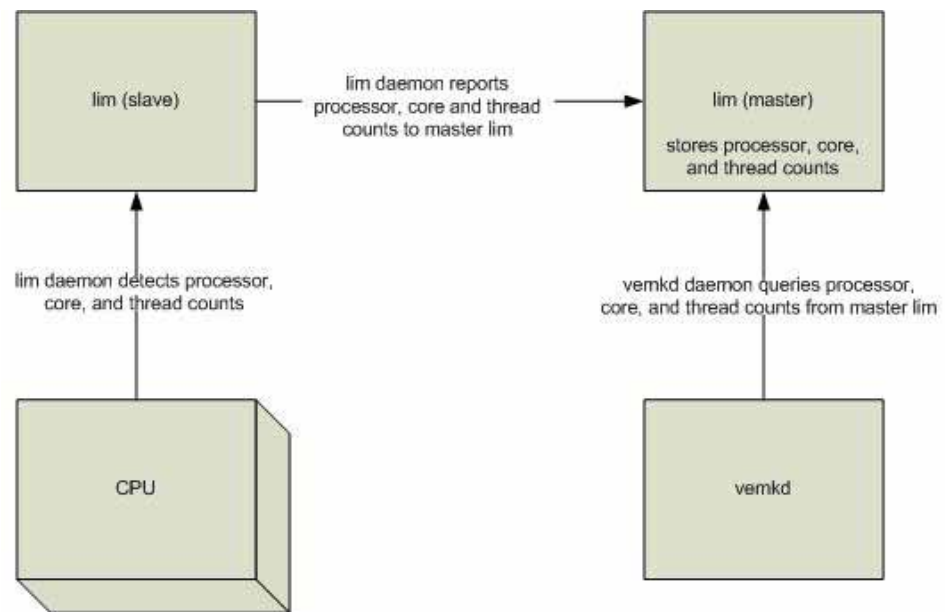
- ◆ The number of processors (this is the `ncpus` default setting)

- ◆ Cores—the number of cores (per processor) * the number of processors
- ◆ Threads—the number of threads (per core) * the number of cores (per processor) * the number of processors

A cluster administrator globally defines how `ncpus` is computed using the `EGO_DEFINE_NCPUS` parameter in `lsf.conf` or `ego.conf` (instead of `LSF_ENABLE_DUALCORE` in `lsf.conf`, or `EGO_ENABLE_DUALCORE` in `ego.conf`).

See [Define ncpus—processors, cores, or threads](#) on page 243 for details.

LIM detects and stores the number of processors, cores, and threads for all supported architectures. The following diagram illustrates the flow of information between daemons, CPUs, and other components.



Although the `ncpus` computation is applied globally, it can be overridden on a per-host basis. See [Override the global configuration of ncpus computation](#) on page 244 for details.

To correctly detect processors, cores, and threads, LIM assumes that all physical processors on a single machine are of the same type.

In cases where CPU architectures and operating system combinations may not support accurate processor, core, thread detection, LIM uses the defaults of 1 processor, 1 core per physical processor, and 1 thread per core. If LIM detects that it is running in a virtual environment (for example, VMware®), each detected processor is similarly reported (as a single-core, single-threaded, physical processor).

LIM only detects hardware that is recognized by the operating system. LIM detection uses processor- or OS-specific techniques (for example, the Intel CPUID instruction, or Solaris `kstat()/core_id`). If the operating system does not

recognize a CPU or core (for example, if an older OS does not recognize a quad-core processor and instead detects it as dual-core), then LIM will not recognize it either.

NOTE: RQL normalization never considers threads. Consider a hyper-thread enabled Pentium: Threads are not full-fledged CPUs, so considering them as CPUs would artificially lower the system load.

ncpus detection on AIX

On a machine running AIX, detection of `ncpus` is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is always the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).

Define `ncpus`—processors, cores, or threads

A cluster administrator must define how `ncpus` is computed. Usually, the number of available job slots is equal to the value of `ncpus`; however, slots can be redefined at the EGO resource group level. The `ncpus` definition is globally applied across the cluster.

1 Open `lsf.conf` or `ego.conf`.

- ◆ **UNIX and Linux:**

`LSF_CONFDIR/lsf.conf`

`LSF_CONFDIR/ego/cluster_name/kernel/conf/ego.conf`

- ◆ **Windows:**

`LSF_CONFDIR\lsf.conf`

`LSF_CONFDIR\ego\cluster_name\kernel\conf\ego.conf`

IMPORTANT: You can set `EGO_DEFINE_NCPUS` in `ego.conf` only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set `EGO_DEFINE_NCPUS` in `lsf.conf`.

2 Define the parameter `EGO_DEFINE_NCPUS=[procs | cores | threads]`.

Set it to one of the following:

- ◆ `procs` (where `ncpus=procs`)
- ◆ `cores` (where `ncpus=procs * cores`)
- ◆ `threads` (where `ncpus=procs * cores * threads`)

By default, `ncpus` is set to `procs` (number of processors).

NOTE: In clusters with older LIMs that do not recognize cores and threads, `EGO_DEFINE_NCPUS` is ignored. In clusters where only the master LIM recognizes cores and threads, the master LIM assigns default values (for example, in Platform LSF 6.2: 1 core, -1 thread).

3 Save and close `lsf.conf` or `ego.conf`.

TIP: As a best practice, set `EGO_DEFINE_NCPUS` instead of `EGO_ENABLE_DUALCORE`. The functionality of `EGO_ENABLE_DUALCORE=y` is preserved by setting `EGO_DEFINE_NCPUS=cores`.

Interaction with `LSF_LOCAL_RESOURCES` in `lsf.conf`

If EGO is enabled, and `EGO_LOCAL_RESOURCES` is set in `ego.conf` and `LSF_LOCAL_RESOURCES` is set in `lsf.conf`, `EGO_LOCAL_RESOURCES` takes precedence.

Override the global configuration of `ncpus` computation

The cluster administrator globally defines how the `ncpus` resource is computed. The `ncpus` global definition can be overridden on specified dynamic and static hosts in the cluster.

Defining computation of `ncpus` on dynamic hosts

1 Open `lsf.conf` or `ego.conf`.

- ◆ **UNIX and Linux:**

`LSF_CONFDIR/lsf.conf`

`LSF_CONFDIR/ego/cluster_name/kernel/conf/ego.conf`

- ◆ **Windows:**

`LSF_CONFDIR\lsf.conf`

`LSF_CONFDIR\ego\cluster_name\kernel\conf\ego.conf`

IMPORTANT: You can set `EGO_LOCAL_RESOURCES` in `ego.conf` only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set `EGO_LOCAL_RESOURCES` in `lsf.conf`.

2 Define the parameter `EGO_LOCAL_RESOURCES="[resource resource_name]"`.

Set `resource_name` to one of the following:

- ◆ `define_ncpus_procs`
- ◆ `define_ncpus_cores`
- ◆ `define_ncpus_threads`

NOTE: Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

- ◆ **Windows:** `EGO_LOCAL_RESOURCES="[type NTX86] [resource define_ncpus_procs]"`
- ◆ **Linux:** `EGO_LOCAL_RESOURCES="[resource define_ncpus_cores]"`

3 Save and close `ego.conf`.

NOTE: In multi-cluster environments, if `ncpus` is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined `ncpus` settings *only* take effect within the cluster for which `EGO_DEFINE_NCPUS` is defined.

Defining computation of `ncpus` on static hosts

1 Open `lsf.cluster.cluster_name`.

- ◆ **Linux:** `LSF_CONFDIR/lsf.cluster.cluster_name`
- ◆ **Windows:** `LSF_CONFDIR\lsf.cluster.cluster_name`

2 Find the host you for which you want to define `ncpus` computation. In the **RESOURCES** column, add one of the following definitions:

- ◆ `define_ncpus_procs`
- ◆ `define_ncpus_cores`
- ◆ `define_ncpus_threads`

NOTE: Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

```
Begin Host
HOSTNAME  model      type      r1m  mem  swp  RESOURCES  #Keywords
#lemon    PC200    LINUX86   3.5  1    2    (linux)
#plum     !         NTX86     3.5  1    2    (nt)
Host_name !         NTX86     -    -    -    (define_ncpus.procs)
End      Host
```

3 Save and close `lsf.cluster.cluster_name`.

4 Restart the master host.

NOTE: In multi-cluster environments, if `ncpus` is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined `ncpus` settings *only* take effect within the cluster for which `EGO_DEFINE_NCPUS` is defined.

Interaction with `LSF_LOCAL_RESOURCES` in `lsf.conf`

If `EGO` is enabled, and `EGO_LOCAL_RESOURCES` is set in `ego.conf` and `LSF_LOCAL_RESOURCES` is set in `lsf.conf`, `EGO_LOCAL_RESOURCES` takes precedence.

Automatic Detection of Hardware Reconfiguration

Some UNIX operating systems support dynamic hardware reconfiguration—that is, the attaching or detaching of system boards in a live system without having to reboot the host.

Supported platforms

LSF is able to recognize changes in `ncpus`, `maxmem`, `maxswp`, `maxtmp` in the following platforms:

- ◆ Sun Solaris 2.5+
- ◆ HP-UX 10.10+
- ◆ IBM AIX 4.0+
- ◆ SGI IRIX 6.2+

Dynamic changes in `ncpus`

LSF is able to automatically detect a change in the number of processors in systems that support dynamic hardware reconfiguration.

The local LIM checks if there is a change in the number of processors at an internal interval of 2 minutes. If it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`. The local LIM then sends this new information to the master LIM.

Dynamic changes in `maxmem`, `maxswp`, `maxtmp`

If you dynamically change `maxmem`, `maxswp`, or `maxtmp` without changing the number of processors, you need to restart the local LIM with the command `lsadmin limrestart` so that it can recognize the changes.

If you dynamically change the number of processors and any of `maxmem`, `maxswp`, or `maxtmp`, the change will be automatically recognized by LSF. When it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`.

Viewing dynamic hardware changes

lsxxx Commands

There may be a 2 minute delay before the changes are recognized by `lsxxx` commands (for example, before `lshosts` displays the changes).

bxxx Commands

There may be at most a 2 + 10 minute delay before the changes are recognized by `bxxx` commands (for example, before `bhosts -l` displays the changes).

This is because `mbatchd` contacts the master LIM at an internal interval of 10 minutes.

Platform MultiCluster

Configuration changes from a local cluster are communicated from the master LIM to the remote cluster at an interval of $2 * \text{CACHE_INTERVAL}$. The parameter `CACHE_INTERVAL` is configured in `lsf.cluster.cluster_name` and is by default 60 seconds.

This means that for changes to be recognized in a remote cluster there is a maximum delay of 2 minutes + $2 * \text{CACHE_INTERVAL}$.

How dynamic hardware changes affect LSF

LSF uses `ncpus`, `maxmem`, `maxswp`, `maxtmp` to make scheduling and load decisions.

When processors are added or removed, LSF licensing is affected because LSF licenses are based on the number of processors.

If you put a processor offline:

- ◆ Per host or per-queue load thresholds may be exceeded sooner. This is because LSF uses the number of CPUs and relative CPU speeds to calculate effective run queue length.
- ◆ The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) increases.
- ◆ Jobs may also be suspended or not dispatched because of load thresholds.
- ◆ Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be exceeded sooner.

If you put a new processor online:

- ◆ Load thresholds may be reached later.
- ◆ The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) is decreased.
- ◆ Jobs suspended due to load thresholds may be resumed.

Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be reached later.

Adding Resources

Contents

- ◆ [About Configured Resources](#) on page 249
- ◆ [Add New Resources to Your Cluster](#) on page 249
- ◆ [Static Shared Resource Reservation](#) on page 253
- ◆ [External Load Indices and ELIM](#) on page 253
- ◆ [Modifying a Built-In Load Index](#) on page 258

About Configured Resources

LSF schedules jobs based on available resources. There are many resources built into LSF, but you can also add your own resources, and then use them same way as built-in resources.

For maximum flexibility, you should characterize your resources clearly enough so that users have satisfactory choices. For example, if some of your machines are connected to both Ethernet and FDDI, while others are only connected to Ethernet, then you probably want to define a resource called `fddi` and associate the `fddi` resource with machines connected to FDDI. This way, users can specify resource `fddi` if they want their jobs to run on machines connected to FDDI.

Add New Resources to Your Cluster

- 1 Log in to any host in the cluster as the LSF administrator.
- 2 Define new resources in the `Resource` section of `lsf.shared`. Specify at least a name and a brief description, which will be displayed to a user by `lsinfo`. See [Configuring lsf.shared Resource Section](#) on page 250.
- 3 For static Boolean resources, for all hosts that have the new resources, add the resource name to the `RESOURCES` column in the `Host` section of `lsf.cluster.cluster_name`.

- 4 For shared resources, for all hosts that have the new resources, associate the resources with the hosts (you might also have a reason to configure non-shared resources in this section).
See [Configuring Isf.cluster.cluster_name ResourceMap Section](#) on page 251.
- 5 Reconfigure your cluster.

Configuring Isf.shared Resource Section

Configured resources are defined in the `Resource` section of `Isf.shared`. There is no distinction between shared and non-shared resources.

You must specify at least a name and description for the resource, using the keywords `RESOURCENAME` and `DESCRIPTION`.

- ◆ A resource name cannot begin with a number.
- ◆ A resource name cannot contain any of the following characters:
: . () [+ - * / ! & | < > @ =
- ◆ A resource name cannot be any of the following reserved keywords:
cpu cpuf io logins ls idle maxmem maxswp maxtmp type model status
it
mem ncpus define_ncpus_cores define_ncpus_procs
define_ncpus_threads ndisks pg r15m r15s r1m swap swp tmp ut
- ◆ To avoid conflict with `inf` and `nan` keywords in 3rd-party libraries, resource names should not begin with `inf` or `nan` (upper case or lower case). Resource requirement strings, such as `-R "infra"` or `-R "nano"` will cause an error. Use `-R "defined(infixx)"` or `-R "defined(nanxx)"`, to specify these resource names.
- ◆ Resource names are case sensitive
- ◆ Resource names can be up to 39 characters in length

You can also specify:

- ◆ The resource type (`TYPE = Boolean | String | Numeric`). The default is Boolean.
- ◆ For dynamic resources, the update interval (`INTERVAL`, in seconds)
- ◆ For numeric resources, where a higher value indicates greater load (`INCREASING = Y`)
- ◆ For numeric shared resources, where LSF releases the resource when a job using the resource is suspended (`RELEASE = Y`)

When the optional attributes are not specified, the resource is treated as static and Boolean.

Example

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
mips          Boolean  ()   ()   (MIPS architecture)
dec           Boolean  ()   ()   (DECStation system)
scratch       Numeric  30   N    (Shared scratch space on server)
synopsys      Numeric  30   N    (Floating licenses for Synopsys)
```

```

verilog      Numeric  30   N      (Floating licenses for Verilog)
console     String   30   N      (User Logged in on console)
End Resource

```

Resources required for JSDL

The following resources are pre-defined to support the submission of jobs using JSDL files.

```

Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
osname       String   600   ()      (OperatingSystemName)
osver        String   600   ()      (OperatingSystemVersion)
cpuarch      String   600   ()      (CPUArchitectureName)
cpuspeed     Numeric  60    Y       (IndividualCPUSpeed)
bandwidth    Numeric  60    Y       (IndividualNetworkBandwidth)
End Resource

```

Configuring `lsf.cluster.cluster_name` ResourceMap Section

Resources are associated with the hosts for which they are defined in the ResourceMap section of `lsf.cluster.cluster_name`.

For each resource, you must specify the name and the hosts that have it.

If the ResourceMap section is not defined, then any dynamic resources specified in `lsf.shared` are not tied to specific hosts, but are shared across all hosts in the cluster.

Example

A cluster consists of hosts `host1`, `host2`, and `host3`.

```

Begin ResourceMap
RESOURCENAME LOCATION
verilog      (5@[all ~host1 ~host2])
synopsys     (2@[host1 host2] 2@[others])
console      (1@[host1] 1@[host2] 1@[host3])
xyz          (1@[default])
End ResourceMap

```

In this example:

- ◆ 5 units of the `verilog` resource are defined on `host3` only (all hosts except `host1` and `host2`).
- ◆ 2 units of the `synopsys` resource are shared between `host1` and `host2`. 2 more units of the `synopsys` resource are defined on `host3` (shared among all the remaining hosts in the cluster).
- ◆ 1 unit of the `console` resource is defined on each host in the cluster (assigned explicitly). 1 unit of the `xyz` resource is defined on each host in the cluster (assigned with the keyword `default`).

Resources required for JSDL

If you plan to submit jobs using JSDL files, you must uncomment the following lines:

RESOURCENAME	LOCATION
<code>osname</code>	<code>[default]</code>
<code>osver</code>	<code>[default]</code>
<code>cpuarch</code>	<code>[default]</code>
<code>cpuspeed</code>	<code>[default]</code>
<code>bandwidth</code>	<code>[default]</code>

RESOURCENAME

The name of the resource, as defined in `lsf.shared`.

LOCATION

Defines the hosts that share the resource. For a static resource, you must define an initial value here as well. Do not define a value for a dynamic resource.

Possible states of a resource:

- ◆ Each host in the cluster has the resource
- ◆ The resource is shared by all hosts in the cluster
- ◆ There are multiple instances of a resource within the cluster, and each instance is shared by a unique subset of hosts.

Syntax

`([resource_value@][host_name... | all [~host_name]... | others | default] ...)`

- ◆ For static resources, you must include the resource value, which indicates the quantity of the resource. Do not specify the resource value for dynamic resources because information about dynamic resources is updated by ELIM.
- ◆ Type square brackets around the list of hosts, as shown. You can omit the parenthesis if you only specify one set of hosts.
- ◆ Each set of hosts within square brackets specifies an instance of the resource. The same host cannot be in more than one instance of a resource. All hosts within the instance share the quantity of the resource indicated by its value.
- ◆ The keyword `all` refers to all the server hosts in the cluster, collectively. Use the not operator (`~`) to exclude hosts or host groups.
- ◆ The keyword `others` refers to all hosts not otherwise listed in the instance.
- ◆ The keyword `default` refers to each host in the cluster, individually.

Non-batch configuration

The following items should be taken into consideration when configuring resources.

- ◆ In `lsf.cluster.cluster_name`, the `Host` section must precede the `ResourceMap` section, since the `ResourceMap` section uses the host names defined in the `Host` section.

- ◆ Use the RESOURCES column in the `Host` section of the `lsf.cluster.cluster_name` file to associate static Boolean resources with particular hosts.
- ◆ Most resources specified in the `ResourceMap` section are interpreted by LSF commands as shared resources, which are displayed using `lsload -s` or `lshosts -s`. The exceptions are:
 - ❖ Non-shared static resources
 - ❖ Dynamic numeric resources specified using the `default` keyword. These are host-based resources and behave like the built-in load indices such as `mem` and `swp`. They are viewed using `lsload -l` or `lsload -L`.

Static Shared Resource Reservation

You must use resource reservation to prevent over-committing static shared resources when scheduling.

The usual situation is that you configure single-user application licenses as static shared resources, and make that resource one of the job requirements. You should also reserve the resource for the duration of the job. Otherwise, LSF updates resource information, assumes that all the static shared resources can be used, and places another job that requires that license. The additional job cannot actually run if the license is already taken by a running job.

If every job that requests a license and also reserves it, LSF updates the number of licenses at the start of each new dispatch turn, subtracts the number of licenses that are reserved, and only dispatches additional jobs if there are licenses available that are not already in use.

Reserving a static shared resource

To indicate that a shared resource is to be reserved while a job is running, specify the resource name in the `rusage` section of the resource requirement string.

Example

You configured licenses for the Verilog application as a resource called `verilog_lic`. To submit a job that will run on a host when there is a license available:

```
bsub -R "select[defined(verilog_lic)] rusage[verilog_lic=1]" myjob
```

If the job can be placed, the license it uses will be reserved until the job completes.

External Load Indices and ELIM

The LSF Load Information Manager (LIM) collects built-in load indices that reflect the load situations of CPU, memory, disk space, I/O, and interactive activities on individual hosts.

While built-in load indices might be sufficient for most jobs, you might have special workload or resource dependencies that require custom *external load indices* defined and configured by the LSF administrator. Load and shared resource information from external load indices, are used the same as built in load indices for job scheduling and host selection.

You can write an External Load Information Manager (ELIM) program that collects the values of configured external load indices and updates LIM when new values are received.

An ELIM can be as simple as a small script, or as complicated as a sophisticated C program. A well-defined protocol allows the ELIM to talk to LIM.

The ELIM executable must be located in `LSF_SERVERDIR`.

- ◆ [How LSF supports multiple ELIMs](#) on page 254
- ◆ [Configuring your application-specific SELIM](#) on page 254
- ◆ [How LSF uses ELIM for external resource collection](#) on page 255
- ◆ [Writing an ELIM](#) on page 256
- ◆ [Setting up an ELIM to support JSDL](#) on page 258
- ◆ [Debugging an ELIM](#) on page 258

How LSF supports multiple ELIMs

To increase LIM reliability, LSF Version 7 supports the configuration of multiple ELIM executables.

Master ELIM (melim)

A master ELIM (`melim`) is installed in `LSF_SERVERDIR`.

`melim` manages multiple site-defined sub-ELIMs (SELIMs) and reports external load information to LIM. `melim` does the following:

- ◆ Starts and stops SELIMs
- ◆ Checks syntax of load information reporting on behalf of LIM
- ◆ Collects load information reported from SELIMs
- ◆ Merges latest valid load reports from each SELIM and sends merged load information back to LIM

ELIM failure

Multiple slave ELIMs managed by a master ELIM increases reliability by protecting LIM:

- ◆ ELIM output is buffered
- ◆ Incorrect resource format or values are checked by ELIM
- ◆ SELIMs are independent of each other; one SELIM hanging while waiting for load information does not affect the other SELIMs

Error logging

MELIM logs its own activities and data into the log file

`LSF_LOGDIR/melim.log.host_name`.

Configuring your application-specific SELIM

The master ELIM is installed as `LSF_SERVERDIR/melim`. After installation:

- 1 Define the external resources you need.
- 2 Write your application-specific SELIM to track these resources, as described in [Writing an ELIM](#) on page 256.
- 3 Put your ELIM in `LSF_SERVERDIR`.

Naming your ELIM

Use the following naming conventions:

- ◆ On UNIX, `LSF_SERVERDIR/elim.application`
For example, `elim.license`
- ◆ On Windows, `LSF_SERVERDIR\elim.application.[exe |bat]`
For example, `elim.license.exe`

Existing ELIMs

TIP: Your existing ELIMs do not need to follow this convention and do not need to be renamed. However, since `melim` invokes any ELIM that follows this convention, you should move any backup copies of your ELIM out of `LSF_SERVERDIR` or choose a name that does not follow the convention (for example, use `elim_bak` instead of `elim.bak`).

`elim.user` is reserved

CAUTION: The name `elim.user` is reserved for backward compatibility. Do not use the name `elim.user` for your application-specific `elim`.

How LSF uses ELIM for external resource collection

The values of static external resources are specified through the `lsf.cluster.cluster_name` configuration file. The values of all dynamic resources, regardless of whether they are shared or host-based, are collected through an ELIM.

When an ELIM is started

An ELIM is started in the following situations:

- ◆ On every host, if any dynamic resource is configured as host-based. For example, if the `LOCATION` field in the `ResourceMap` section of `lsf.cluster.cluster_name` is `([default])`, then every host will start an ELIM.
- ◆ On the master host, for any cluster-wide resources. For example, if the `LOCATION` field in the `ResourceMap` section of `lsf.cluster.cluster_name` is `([all])`, then an ELIM is started on the master host.
- ◆ On the first host specified for each instance, if multiple instances of the resource exist within the cluster. For example, if the `LOCATION` field in the `ResourceMap` section of `lsf.cluster.cluster_name` is `([hostA hostB hostC] [hostD hostE hostF])`, then an ELIM will be started on `hostA` and `hostD` to report the value of that resource for that set of hosts.

If the host reporting the value for an instance goes down, then an ELIM is started on the next available host in the instance. In above example, if `hostA` became unavailable, an ELIM is started on `hostB`. If the `hostA` becomes available again then the ELIM on `hostB` is shut down and the one on `hostA` is started.

There is only one ELIM on each host, regardless of the number of resources on which it reports. If only cluster-wide resources are to be collected, then an ELIM will only be started on the master host.

Environment variables

When LIM starts, the following environment variables are set for ELIM:

- ◆ `LSF_MASTER`: This variable is defined if the ELIM is being invoked on the master host. It is undefined otherwise. This can be used to test whether the ELIM should report on cluster-wide resources that only need to be collected on the master host.

- ◆ **LSF_RESOURCES:** This variable contains a list of resource names (separated by spaces) on which the ELIM is expected to report. A resource name is only put in the list if the host on which the ELIM is running shares an instance of that resource.

Writing an ELIM

The ELIM must be an executable program, either an interpreted script or compiled code.

ELIM output

The ELIM communicates with the LIM by periodically writing a load update string to its standard output. The load update string contains the number of indices followed by a list of name-value pairs in the following format:

```
number_indices [index_name index_value]...
```

For example,

```
3 tmp2 47.5 nio 344.0 licenses 5
```

This string reports three indices: `tmp2`, `nio`, and `licenses`, with values 47.5, 344.0, and 5 respectively. Index values must be numbers between `-INFINIT_LOAD` and `INFINIT_LOAD` as defined in the `lsf.h` header file.

If the ELIM is implemented as a C program, as part of initialization it should use `setbuf(3)` to establish unbuffered output to `stdout`.

The ELIM should ensure that the entire load update string is written successfully to `stdout`. This can be done by checking the return value of `printf(3s)` if the ELIM is implemented as a C program or as the return code of `/bin/echo(1)` from a shell script. The ELIM should exit if it fails to write the load information.

Each LIM sends updated load information to the master every 15 seconds. Depending on how quickly your external load indices change, the ELIM should write the load update string at most once every 15 seconds. If the external load indices rarely change, the ELIM can write the new values only when a change is detected. The LIM continues to use the old values until new values are received.

ELIM location

The executable for the ELIM must be in `LSF_SERVERDIR`.

Use the following naming conventions:

- ◆ On UNIX, `LSF_SERVERDIR/elim.application`
For example, `elim.license`
- ◆ On Windows, `LSF_SERVERDIR\elim.application.[exe |bat]`
For example, `elim.license.exe`

If LIM expects some resources to be collected by an ELIM according to configuration, it invokes the ELIM automatically on startup. The ELIM runs with the same user ID and file access permission as the LIM.

ELIM restart

The MELIM restarts the ELIM if it exits; to prevent problems in case of a fatal error in the ELIM, it is restarted at most once every 90 seconds. When the LIM terminates, it sends a `SIGTERM` signal to the ELIM. The ELIM must exit upon receiving this signal.

Example

1 Write an ELIM.

The following sample ELIM (LSF_SERVERDIR/elim.myrc) sets the value of myrc resource to 2. In a real ELIM, you would have a command to retrieve whatever value you want to retrieve and set the value.

```
#!/bin/sh
while :
do
  # set the value for resource "myrc"
  val=2

  # create an output string in the format:
  # number_indices index1_name index1_value...

  reportStr="1 myrc $val"
  echo "$reportStr"

  # wait for 30 seconds before reporting again
  sleep 30
done
```

2 Test this ELIM by running it from the command line.

```
./elim.myrc
```

It should give you the output:

```
1 myrc 2
```

3 Copy the ELIM to LSF_SERVERDIR and make sure it has the name elim.myrc.

4 Define the myrc resource in lsf.shared.

In this case, we are defining the resource as Numeric because we want it to accept numbers. The value does not increase with load.

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
myrc Numeric 30 N (custom resource to trigger elim to start up)
End Resource
```

5 Map the myrc resource to hosts in lsf.cluster.*cluster_name*. In this case, we want this resource to reside only on hostA.

```
Begin ResourceMap
RESOURCENAME LOCATION
myrc [hostA]
End ResourceMap
```

6 Reconfigure the cluster:

- a Run `lsadmin reconfig`.
- b Run `badmin mbdrestart`.

7 Display the resource with the command `lsload -l`. You should be able to see the new resource and value:

```
HOST_NAME status r15s r1m r15m ut pg io ls it tmp swp mem myrc
hostA ok 0.4 0.4 0.4 0% 0.0 0 22 0 24M 26M 6M 2
```

Additional examples

Example code for an ELIM is included in the `LSF_MISC/examples` directory. The `elim.c` file is an ELIM written in C. You can modify this example to collect the load indices you want.

Setting up an ELIM to support JSDL

To support the use of JSDL files at job submission, LSF collects the following load metrics:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth (This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources. To enable the use of `elim.jsdl`, uncomment the lines for these resources in the ResourceMap section of the file `lsf.cluster.clustername`.

Debugging an ELIM

Set the parameter `LSF_ELIM_DEBUG=y` in the Parameters section of `lsf.cluster.cluster_name` to log all load information received by LIM from the ELIM in the LIM log file.

Set the parameter `LSF_ELIM_BLOCKTIME=seconds` in the Parameters section of `lsf.cluster.cluster_name` to configure how long MELIM waits before restarting the ELIM.

Use the parameter `LSF_ELIM_RESTARTS=integer` in the Parameters section of `lsf.cluster.cluster_name` to limit the number of times an ELIM can be restarted.

See the *Platform LSF Configuration Reference* for more details on these parameters.

Modifying a Built-In Load Index

The ELIM can return values for the built-in load indices. In this case the value produced by the ELIM overrides the value produced by the LIM.

Considerations

- ◆ The ELIM must ensure that the semantics of any index it supplies are the same as that of the corresponding index returned by the `lsinfo(1)` command.
- ◆ The name of an external load index must not be one of the resource name aliases: `cpu`, `idle`, `login`, or `swap`. To override one of these indices, use its formal name: `r1m`, `it`, `ls`, or `swp` as the ELIM output.
- ◆ You must configure an external load index in `lsf.shared` even if you are overriding a built-in load index.

Override a load index

Some sites prefer to use `/usr/tmp` for temporary files. You can override the `tmp` load index.

- 1 Write a program that periodically measures the space in the `/usr/tmp` file system and writes the value to standard output. For details on format, see [Writing an ELIM](#) on page 256.

For example, the program writes to its standard output:

```
1 tmp 47.5
```

- 2 Name the program `elim` and store it in the `LSF_SERVERDIR` directory. All default load indices are local resources, so the `elim` must run locally on every machine.

- 3 Define the resource.

Since the name of built-in load indices is not allowed in `lsf.shared`, define a custom resource to trigger the `elim`.

For example:

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
my_tmp Numeric 30 N (custom resource to trigger elim to start up)
End Resource
```

- 4 Map the resource to hosts in `lsf.cluster.cluster_name`.

- ❖ To override the `tmp` load index on every host, specify the keyword `default`:

```
Begin ResourceMap
RESOURCENAME LOCATION
my_tmp [default]
End ResourceMap
```

- ❖ To override the `tmp` load index only on specific hosts, specify the host names:

```
Begin ResourceMap
RESOURCENAME LOCATION
my_tmp ([host1][host2][host3])
End ResourceMap
```

Modifying a Built-In Load Index

Managing Software Licenses with LSF

Software licenses are valuable resources that must be fully utilized. This section discusses how LSF can help manage licensed applications to maximize utilization and minimize job failure due to license problems.

Contents

- ◆ [Using Licensed Software with LSF](#) on page 261
- ◆ [Host-locked Licenses](#) on page 261
- ◆ [Counted Host-Locked Licenses](#) on page 261
- ◆ [Network Floating Licenses](#) on page 262

Using Licensed Software with LSF

Many applications have restricted access based on the number of software licenses purchased. LSF can help manage licensed software by automatically forwarding jobs to licensed hosts, or by holding jobs in batch queues until licenses are available.

Host-locked Licenses

Host-locked software licenses allow users to run an unlimited number of copies of the product on each of the hosts that has a license.

Configuring host-locked licenses

You can configure a Boolean resource to represent the software license, and configure your application to require the license resource. When users run the application, LSF chooses the best host from the set of licensed hosts.

See [Boolean resources](#) on page 232 for information about configuring Boolean resources.

See the *Platform LSF Configuration Reference* for information about the `lsf.task` file and instructions on configuring resource requirements for an application.

Counted Host-Locked Licenses

Counted host-locked licenses are only available on specific licensed hosts, but also place a limit on the maximum number of copies available on the host.

Configuring counted host-locked licenses

You configure counted host-locked licenses by having LSF determine the number of licenses currently available. Use either of the following to count the host-locked licenses:

- ◆ External LIM (ELIM)
- ◆ A `check_licenses` shell script

Using an External LIM (ELIM)

To use an external LIM (ELIM) to get the number of licenses currently available, configure an external load index `licenses` giving the number of free licenses on each host. To restrict the application to run only on hosts with available licenses, specify `licenses>=1` in the resource requirements for the application.

See [External Load Indices and ELIM](#) on page 253 for instructions on writing and using an ELIM and configuring resource requirements for an application.

See the *Platform LSF Configuration Reference* for information about the `lsf.task` file.

Using a `check_license` script

There are two ways to use a `check_license` shell script to check license availability and acquire a license if one is available:

- ◆ Configure the `check_license` script as a job-level pre-execution command when submitting the licensed job:
- ```
bsub -m licensed_hosts -E check_license licensed_job
```
- ◆ Configure the `check_license` script as a queue-level pre-execution command. See [Configuring Pre- and Post-Execution Commands](#) on page 531 for information about configuring queue-level pre-execution commands.

It is possible that the license becomes unavailable between the time the `check_license` script is run, and when the job is actually run. To handle this case, configure a queue so that jobs in this queue will be requeued if they exit with values indicating that the license was not successfully obtained.

See [Automatic Job Requeue](#) on page 441 for more information.

## Network Floating Licenses

A network floating license allows a fixed number of machines or users to run the product at the same time, without restricting which host the software can run on. Floating licenses are cluster-wide resources; rather than belonging to a specific host, they belong to all hosts in the cluster.

LSF can be used to manage floating licenses using the following LSF features:

- ◆ Shared resources
- ◆ Resource reservation
- ◆ Job queuing

Using LSF to run licensed software can improve the utilization of the licenses. The licenses can be kept in use 24 hours a day, 7 days a week. For expensive licenses, this increases their value to the users. Floating licenses also increase productivity, because users do not have to wait for a license to become available.

LSF jobs can make use of floating licenses when:

- ◆ All license jobs are run through LSF
- ◆ Licenses are managed outside of LSF control

## All licenses used through LSF

If all jobs requiring licenses are submitted through LSF, then LSF could regulate the allocation of licenses to jobs and ensure that a job is not started if the required license is not available. A static resource is used to hold the total number of licenses that are available. The static resource is used by LSF as a counter which is decremented by the resource reservation mechanism each time a job requiring that resource is started.

### Example

For example, suppose that there are 10 licenses for the Verilog package shared by all hosts in the cluster. The LSF configuration files should be specified as shown below. The resource is a static value, so an ELIM is not necessary.

#### lsf.shared

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
verilog Numeric () N (Floating licenses for
Verilog)
End Resource
```

#### lsf.cluster.cluster\_name

```
Begin ResourceMap
RESOURCENAME LOCATION
verilog (10@[all])
End ResourceMap
```

#### Submitting jobs

The users would submit jobs requiring verilog licenses as follows:

```
bsub -R "rusage[verilog=1]" myprog
```

## Licenses used outside of LSF control

To handle the situation where application licenses are used by jobs outside of LSF, use an ELIM to dynamically collect the actual number of licenses available instead of relying on a statically configured value. The ELIM periodically informs LSF of the number of available licenses, and LSF takes this into consideration when scheduling jobs.

### Example

Assuming there are a number of licenses for the Verilog package that can be used by all the hosts in the cluster, the LSF configuration files could be set up to monitor this resource as follows:

#### lsf.shared

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
verilog Numeric 60 N (Floating licenses
for Verilog)
End Resource
```

### `lsf.cluster.cluster_name`

```
Begin ResourceMap
RESOURCENAME LOCATION
verilog ([all])
End ResourceMap
```

The `INTERVAL` in the `lsf.shared` file indicates how often the ELIM is expected to update the value of the `Verilog` resource—in this case every 60 seconds. Since this resource is shared by all hosts in the cluster, the ELIM only needs to be started on the master host. If the `Verilog` licenses can only be accessed by some hosts in the cluster, specify the `LOCATION` field of the `ResourceMap` section as (`[hostA hostB hostC ...]`). In this case an ELIM is only started on `hostA`.

### Submitting jobs

The users would submit jobs requiring `verilog` licenses as follows:

```
bsub -R "rusage[verilog=1:duration=1]" myprog
```

### Configuring a dedicated queue for floating licenses

Whether you run all license jobs through LSF or run jobs that use licenses that are outside of LSF control, you can configure a dedicated queue to run jobs requiring a floating software license.

For each job in the queue, LSF reserves a software license before dispatching a job, and releases the license when the job finishes.

Use the `bhosts -s` command to display the number of licenses being reserved by the dedicated queue.

### Example

The following example defines a queue named `q_verilog` in `lsb.queues` dedicated to jobs that require `Verilog` licenses:

```
Begin Queue
QUEUE_NAME = q_verilog
RES_REQ=rusage[verilog=1:duration=1]
End Queue
```

The queue named `q_verilog` contains jobs that will reserve one `Verilog` license when it is started.

If the `Verilog` licenses are not cluster-wide, but can only be used by some hosts in the cluster, the resource requirement string should include the `defined()` tag in the `select` section:

```
select[defined(verilog)] rusage[verilog=1]
```

### Preventing underutilization of licenses

One limitation to using a dedicated queue for licensed jobs is that if a job does not actually use the license, then the licenses will be under-utilized. This could happen if the user mistakenly specifies that their application needs a license, or submits a non-licensed job to a dedicated queue.

LSF assumes that each job indicating that it requires a `Verilog` license will actually use it, and simply subtracts the total number of jobs requesting `Verilog` licenses from the total number available to decide whether an additional job can be dispatched.



Use the `duration` keyword in the queue resource requirement specification to release the shared resource after the specified number of minutes expires. This prevents multiple jobs started in a short interval from over-using the available licenses. By limiting the duration of the reservation and using the actual license usage as reported by the ELIM, underutilization is also avoided and licenses used outside of LSF can be accounted for.

## When interactive jobs compete for licenses

In situations where an interactive job outside the control of LSF competes with batch jobs for a software license, it is possible that a batch job, having reserved the software license, may fail to start as its license is intercepted by an interactive job. To handle this situation, configure job requeue by using the `REQUEUE_EXIT_VALUES` parameter in a queue definition in `lsb.queues`. If a job exits with one of the values in the `REQUEUE_EXIT_VALUES`, LSF will requeue the job.

### Example

Jobs submitted to the following queue will use Verilog licenses:

```
Begin Queue
QUEUE_NAME = q_verilog
RES_REQ=rusage[verilog=1:duration=1]
application exits with value 99 if it fails to get license
REQUEUE_EXIT_VALUES = 99
JOB_STARTER = lic_starter
End Queue
```

All jobs in the queue are started by the job starter `lic_starter`, which checks if the application failed to get a license and exits with an exit code of 99. This causes the job to be requeued and LSF will attempt to reschedule it at a later time.

### lic\_starter job starter script

The `lic_starter` job starter can be coded as follows:

```
#!/bin/sh
lic_starter: If application fails with no license, exit 99,
otherwise, exit 0. The application displays
"no license" when it fails without license available.
$* 2>&1 | grep "no license"
if [$? != "0"]
then
 exit 0 # string not found, application got the license
else
 exit 99
fi
```

## For more information

- ◆ See [Automatic Job Requeue](#) on page 441 for more information about configuring job requeue
- ◆ See Chapter 36, “[Job Starters](#)” for more information about LSF job starters



## Job Scheduling Policies

- ◆ [Time Syntax and Configuration](#) on page 269
- ◆ [Deadline Constraint and Exclusive Scheduling](#) on page 275
- ◆ [Preemptive Scheduling](#) on page 277
- ◆ [Specifying Resource Requirements](#) on page 279
- ◆ [Fairshare Scheduling](#) on page 295
- ◆ [Goal-Oriented SLA-Driven Scheduling](#) on page 327
- ◆ [Working with Application Profiles](#) on page 357



## Time Syntax and Configuration

### Contents

- ◆ [Specifying Time Values](#) on page 269
- ◆ [Specifying Time Windows](#) on page 269
- ◆ [Specifying Time Expressions](#) on page 270
- ◆ [Using Automatic Time-based Configuration](#) on page 271

### Specifying Time Values

To specify a time value, a specific point in time, specify at least the hour. Day and minutes are optional.

#### Time value syntax

*time = hour | hour:minute | day:hour:minute*

#### *hour*

Integer from 0 to 23, representing the hour of the day.

#### *minute*

Integer from 0 to 59, representing the minute of the hour.

If you do not specify the minute, LSF assumes the first minute of the hour (:00).

#### *day*

Integer from 0 to 7, representing the day of the week, where 0 represents every day, 1 represents Monday, and 7 represents Sunday.

If you do not specify the day, LSF assumes every day. If you do specify the day, you must also specify the minute.

### Specifying Time Windows

To specify a time window, specify two time values separated by a hyphen (-), with no space in between.

*time\_window = begin\_time-end\_time*

#### Time format

Times are specified in the format:

[*day*:]hour[:*minute*]

where all fields are numbers with the following ranges:

- ◆ *day of the week*: 0-6 (0 is Sunday)
- ◆ *hour*: 0-23
- ◆ *minute*: 0-59

Specify a time window one of the following ways:

- ◆ *hour-hour*
- ◆ *hour:minute-hour:minute*
- ◆ *day:hour:minute-day:hour:minute*

The default value for minute is 0 (on the hour); the default value for day is every day of the week.

You must specify at least the hour. Day of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (:00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

You can specify multiple time windows, but they cannot overlap. For example:

```
timeWindow(8:00-14:00 18:00-22:00)
```

is correct, but

```
timeWindow(8:00-14:00 11:00-15:00)
```

is not valid.

## Examples of time windows

### Daily window

To specify a daily window omit the day field from the time window. Use either the *hour-hour* or *hour:minute-hour:minute* format. For example, to specify a daily 8:30 a.m. to 6:30 p.m window:

```
8:30-18:30
```

### Overnight window

To specify an overnight window make *time1* greater than *time2*. For example, to specify 6:30 p.m. to 8:30 a.m. the following day:

```
18:30-8:30
```

### Weekend window

To specify a weekend window use the day field. For example, to specify Friday at 6:30 p.m to Monday at 8:30 a.m.:

```
5:18:30-1:8:30
```

## Specifying Time Expressions

Time expressions use time windows to specify when to change configurations.

### Time expression syntax

A time expression is made up of the `time` keyword followed by one or more space-separated time windows enclosed in parenthesis. Time expressions can be combined using the `&&`, `|`, and `!` logical operators.

The syntax for a time expression is:

```
expression = time(time_window[time_window ...])
 | expression && expression
 | expression || expression
 | !expression
```

### Example

Both of the following expressions specify weekends (Friday evening at 6:30 p.m. until Monday morning at 8:30 a.m.) and nights (8:00 p.m. to 8:30 a.m. daily).

```
time(5:18:30-1:8:30 20:00-8:30)
time(5:18:30-1:8:30) || time(20:00-8:30)
```

## Using Automatic Time-based Configuration

Variable configuration is used to automatically change LSF configuration based on time windows. It is supported in the following files:

- ◆ lsb.hosts
- ◆ lsb.params
- ◆ lsb.queues
- ◆ lsb.resources
- ◆ lsb.users
- ◆ lsf.licensescheduler

You define automatic configuration changes in configuration files by using if-else constructs and time expressions. After you change the files, reconfigure the cluster with the `badadmin reconfig` command.

The expressions are evaluated by LSF every 10 minutes based on `mbatchd` start time. When an expression evaluates true, LSF dynamically changes the configuration based on the associated configuration statements. Reconfiguration is done in real time without restarting `mbatchd`, providing continuous system availability.

In the following examples, the `#if`, `#else`, `#endif` are not interpreted as comments by LSF but as if-else constructs.

### lsb.hosts example

```
Begin Host
HOST_NAME r15s r1m pg
host1 3/5 3/5 12/20
#if time(5:16:30-1:8:30 20:00-8:30)
host2 3/5 3/5 12/20
#else
host2 2/3 2/3 10/12
#endif
host3 3/5 3/5 12/20
End Host
```

### lsb.params example

```
if 18:30-19:30 is your short job express period, but
you want all jobs going to the short queue by default
and be subject to the thresholds of that queue
```

## Using Automatic Time-based Configuration

```
for all other hours, normal is the default queue

#if time(18:30-19:30)
DEFAULT_QUEUE=short
#else
DEFAULT_QUEUE=normal
#endif
```

## lsb.queues example

```
Begin Queue
...
#if time(8:30-18:30)
 INTERACTIVE = ONLY # interactive only during day shift
#endif
...
End Queue
```

## lsb.resources example

```
Example: limit usage of hosts in 'license1' group and time based
configuration
- 10 jobs can run from normal queue
- any number can run from short queue between 18:30 and 19:30
- all other hours you are limited to 100 slots in the short queue

- each other queue can run 30 jobs

Begin Limit
PER_QUEUE HOSTS SLOTS # Example
normal license1 10
if time(18:30-19:30)
short license1 -
#else
short license1 100
#endif
(all ~normal ~short) license1 30
End Limit
```

## lsb.users example

From 12 - 1 p.m. daily, user `smith` has 10 job slots, but during other hours, user has only 5 job slots.

```
Begin User
USER_NAME MAX_JOBS JL/P
#if time (12-13)
smith 10 -
#else
smith 5 -
```



```

default 1 -
#endif
End User

```

## lsf.licensescheduler example

```

Begin Feature
NAME = f1
#if time(5:16:30-1:8:30 20:00-8:30)
DISTRIBUTION=Lan(P1 2/5 P2 1)
#elif time(3:8:30-3:18:30)
DISTRIBUTION=Lan(P3 1)
#else
DISTRIBUTION=Lan(P1 1 P2 2/5)
#endif
End Feature

```

## Creating if-else constructs

The if-else construct can express single decisions and multi-way decisions by including elif statements in the construct.

### if-else

The syntax for constructing if-else expressions is:

```

#if time(expression)
statement
#else
statement
#endif

```

The `#endif` part is mandatory and the `#else` part is optional.

For syntax of a time expression, see [Specifying Time Expressions](#) on page 270.

### elif

The `#elif` expressions are evaluated in order. If any expression is true, the associated statement is used, and this terminates the whole chain.

The `#else` part handles the default case where none of the other conditions are satisfied.

When you use `#elif`, the `#else` and `#endif` parts are mandatory.

```

#if time(expression)
statement
#elif time(expression)
statement
#elif time(expression)
statement
#else
statement
#endif

```

## Verify configuration

- 1 Use the following LSF commands to verify configuration:

- ❖ `bhosts`

## Using Automatic Time-based Configuration

- ❖ `bladmin ckconfig`
  - ❖ `blimits -c`
  - ❖ `blinfo`
  - ❖ `blstat`
  - ❖ `bparams`
  - ❖ `bqueues`
  - ❖ `bresources`
  - ❖ `busers`
-

## Deadline Constraint and Exclusive Scheduling

### Contents

- ◆ [Using Deadline Constraint Scheduling](#) on page 275
- ◆ [Using Exclusive Scheduling](#) on page 276

### Using Deadline Constraint Scheduling

Deadline constraints will suspend or terminate running jobs at a certain time. There are two kinds of deadline constraints:

- ◆ A run window, specified at the queue level, suspends a running job
- ◆ A termination time, specified at the job level (`bsub -t`), terminates a running job

### Time-based resource usage limits

- ◆ A CPU limit, specified at job or queue level, terminates a running job when it has used up a certain amount of CPU time.
- ◆ A run limit, specified at the job or queue level, terminates a running job after it has spent a certain amount of time in the RUN state.

### How deadline constraint scheduling works

If deadline constraint scheduling is enabled, LSF will not place a job that will be interrupted by a deadline constraint before its run limit expires, or before its CPU limit expires, if the job has no run limit. In this case, deadline constraint scheduling could prevent a job from ever starting. If a job has neither a run limit nor a CPU limit, deadline constraint scheduling has no effect.

A job that cannot start because of a deadline constraint causes an email to be sent to the job owner.

Deadline constraint scheduling only affects the placement of jobs. Once a job starts, if it is still running at the time of the deadline, it will be suspended or terminated because of the deadline constraint or resource usage limit.

## Disabling deadline constraint scheduling

Deadline constraint scheduling is enabled by default. To disable it for a queue, set `IGNORE_DEADLINE=y` in `lsb.queues`.

### Example

LSF will schedule jobs in the `liberal` queue without observing the deadline constraints.

```
Begin Queue
QUEUE_NAME = liberal
IGNORE_DEADLINE=y
End Queue
```

## Using Exclusive Scheduling

Exclusive scheduling gives a job exclusive use of the host that it runs on. LSF dispatches the job to a host that has no other jobs running, and does not place any more jobs on the host until the exclusive job is finished.

### How exclusive scheduling works

When an exclusive job (`bsub -x`) is submitted to an exclusive queue (`EXCLUSIVE=Y` in `lsb.queues`) and dispatched to a host, LSF locks the host (`lockU` status) until the job finishes.

LSF cannot place an exclusive job unless there is a host that has no jobs running on it.

To make sure exclusive jobs can be placed promptly, configure some hosts to run one job at a time. Otherwise, a job could wait indefinitely for a host in a busy cluster to become completely idle.

### Configure an exclusive queue

- 1 To configure an exclusive queue, set `EXCLUSIVE` in the queue definition (`lsb.queues`) to `Y`.

### Configure a host to run one job at a time

- 1 To make sure exclusive jobs can be placed promptly, configure some single-processor hosts to run one job at a time. To do so, set `SLOTS=1` and `HOSTS=all` in `lsb.resources`.

### Submit an exclusive job

- 1 To submit an exclusive job, use the `-x` option of `bsub` and submit the job to an exclusive queue.

## Preemptive Scheduling

### Contents

- ◆ [About Preemptive Scheduling](#) on page 277

### About Preemptive Scheduling

Preemptive scheduling lets a pending high-priority job take job slots away from a running job of lower priority. When two jobs compete for the same job slots, LSF automatically suspends the low-priority job to make slots available to the high-priority job. The low-priority job is resumed as soon as possible.

Use preemptive scheduling if you have long-running low-priority jobs causing high-priority jobs to wait an unacceptably long time.

For detailed information about the preemptive scheduling feature and how to configure it, see the *Platform LSF Configuration Guide*.

### Limitation

The following types of jobs cannot be preempted:

- ◆ Jobs that have been forced to run with the command `brun`
- ◆ NQS jobs

### Preemptive and preemptable queues

**Preemptive queues** Jobs in a preemptive queue can preempt jobs in any queue of lower priority, even if the low-priority queues are not specified as preemptable.

**Preemptable queues** Jobs in a preemptable queue can be preempted by jobs from any queue of a higher priority, even if the high-priority queues are not specified as preemptive.

### Preemptive and preemptable jobs

**Preemptive jobs** Preemptive jobs are pending in a high-priority queue and require the specified job slots. Their queue must be able to preempt the low-priority queue.

## About Preemptive Scheduling

### Preemptable jobs

Preemptable jobs are running in a low-priority queue and are holding the specified job slot. Their queue must be able to be preempted by the high-priority queue.

## Specifying Resource Requirements

### Contents

- ◆ [About Resource Requirements](#) on page 279
- ◆ [Queue-level Resource Requirements](#) on page 280
- ◆ [Job-level Resource Requirements](#) on page 281
- ◆ [About Resource Requirement Strings](#) on page 282
- ◆ [Selection String](#) on page 284
- ◆ [Order String](#) on page 287
- ◆ [Usage String](#) on page 287
- ◆ [Span String](#) on page 291
- ◆ [Same String](#) on page 292

### About Resource Requirements

Resource requirements define which hosts a job can run on. Each job has its resource requirements. Hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it uses the load index values of all the candidate hosts. The load values for each host are compared to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

By default, if a job has no resource requirements, LSF places it on a host of the same type as the submission host (i.e., `type==local`). However, if a job has string or Boolean resource requirements specified and the host type has not been specified, LSF places the job on any host (i.e., `type==any`) that satisfies the resource requirements.

To override the LSF defaults, specify resource requirements explicitly. Resource requirements can be set for queues, for application profiles, or for individual jobs.

## Queue-level Resource Requirements

To best place a job with optimized performance, resource requirements can be specified for each application. This way, you do not have to specify resource requirements every time you submit a job. The LSF administrator may have already configured the resource requirements for your jobs, or you can put your executable name together with its resource requirements into your personal remote task list.

The `bsub` command automatically uses the resource requirements of the job from the remote task lists.

A resource requirement is an expression that contains resource names and operators.

## Resource requirements in application profiles

See Chapter 20, “[Working with Application Profiles](#)” for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

## Queue-level Resource Requirements

Each queue can define resource requirements that will be applied to all the jobs in the queue.

When resource requirements are specified for a queue, and no job-level or application profile resource requirement is specified, the queue-level resource requirements become the default resource requirements for the job.

Resource requirements determined by the queue no longer apply to a running job after running `badmin reconfig`. For example, if you change the `RES_REQ` parameter in a queue and reconfigure the cluster, the previous queue-level resource requirements for running jobs are lost.

### Syntax

The condition for dispatching a job to a host can be specified through the queue-level `RES_REQ` parameter in the queue definition in `lsb.queues`.

### Examples

```
RES_REQ=select[((type==LINUX2.4 && r1m < 2.0)|| (type==AIX && r1m < 1.0))]
```

This allows a queue, which contains LINUX2.4 and AIX hosts, to have different thresholds for different types of hosts.

```
RES_REQ=select[((hname==hostA && mem > 50)|| (hname==hostB && mem > 100))]
```

Using the `hname` resource in the resource requirement string allows you to set up different conditions for different hosts in the same queue.

### Load thresholds

Load thresholds can be configured by your LSF administrator to schedule jobs in queues. Load thresholds specify a load index value.

### loadSched

The scheduling threshold that determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job is not started on the host. This threshold is also used as the condition for resuming suspended jobs.



## loadStop

The suspending condition that determines when running jobs should be suspended.

Thresholds can be configured for each queue, for each host, or a combination of both. To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (>) or less than (<), depending on the load index.

See Chapter 34, “[Load Thresholds](#)” for information about suspending conditions and configuring load thresholds.

## View queue-level resource requirements

- 1 Use `bqueues -l` to view resource requirements (RES\_REQ) defined for the queue:

```
bqueues -l normal
```

```
QUEUE: normal
```

```
-- No description provided. This is the default queue.
```

```
...
```

```
RES_REQ: select[type==any]
```

```
rusage[mem=10,dynamic_rsrc=10:duration=2:decay=1]
```

```
...
```

## Job-level Resource Requirements

Each job can specify resource requirements. Job-level resource requirements override any resource requirements specified in the remote task list.

In some cases, the queue specification sets an upper or lower bound on a resource. If you attempt to exceed that bound, your job will be rejected.

## Syntax

To specify resource requirements for your job, use `bsub -R` and specify the resource requirement string as usual. You can specify multiple `-R` order, same, `rusage`, and `select` sections.

**TIP:** With `esub`, you must use the `&&` operator to specify multiple resource requirements. The `LSB_SUB_RES_REQ` variable in `esub` does not support the use of multiple `-R` sections.

## Examples

```
bsub -R "swp > 15 && hpux order[ut]" myjob
```

or

```
bsub -R "select[swp > 15]" -R "select[hpux] order[ut]" myjob
```

This runs `myjob` on an HP-UX host that is lightly loaded (CPU utilization) and has at least 15 MB of swap memory available.

```
bsub -R "select[swp > 15]" -R "select[hpux] order[r15m]"
-R "order[r15m]" -R rusage[mem=100]" -R "order[ut]" -R "same[type]"
-R "rusage[tmp=50:duration=60]" -R "same[model]" myjob
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

## View job-level resource requirements

- 1 Use `bjobs -l` to view resource requirements defined for the job:

```
bsub -R "type==any" -q normal myjob
Job <2533> is submitted to queue <normal>.

bjobs -l 2533
Job <2533>, User <user1>, Project <default>, Status <DONE>, Queue
<normal>,
 Command <myjob>
Fri May 10 17:21:26: Submitted from host <hostA>, CWD <${HOME}>,
Requested Resources <type==any>;
Fri May 10 17:21:31: Started on <hostB>, Execution Home
</home/user1>, Execution CWD </home/user1>;
Fri May 10 17:21:47: Done successfully. The CPU time used is 0.3
seconds.
...
```

- 2 After a job is finished, use `bhist -l` to view resource requirements defined for the job:

```
bhist -l 2533

Job <2533>, User <user1>, Project <default>, Command <myjob>
Fri May 10 17:21:26: Submitted from host <hostA>, to Queue
<normal>, CWD
 <${HOME}>, Requested Resources <type==any>;
Fri May 10 17:21:31: Dispatched to <hostB>;
Fri May 10 17:21:32: Starting (Pid 1850232);
Fri May 10 17:21:33: Running with execution home </home/user1>,
Execution
 CWD </home/user1>, Execution Pid <1850232>;
Fri May 10 17:21:45: Done successfully. The CPU time used is 0.3
seconds;
...
```

**NOTE:** If you submitted a job with multiple select strings using the `bsub -R` option, `bjobs -l` and `bhist -l` display a single, merged select string.

## About Resource Requirement Strings

Most LSF commands accept a `-R res_req` argument to specify resource requirements. The exact behaviour depends on the command. For example, specifying a resource requirement for the `lsload` command displays the load levels for all hosts that have the requested resources.

Specifying resource requirements for the `lsrun` command causes LSF to select the best host out of the set of hosts that have the requested resources.

A resource requirement string describes the resources a job needs. LSF uses resource requirements to select hosts for remote execution and job execution.

## Resource requirement string sections

- ◆ A selection section (`select`). The selection section specifies the criteria for selecting hosts from the system.
- ◆ An ordering section (`order`). The ordering section indicates how the hosts that meet the selection criteria should be sorted.
- ◆ A resource usage section (`rusage`). The resource usage section specifies the expected resource consumption of the task.
- ◆ A job spanning section (`span`). The job spanning section indicates if a parallel batch job should span across multiple hosts.
- ◆ A same resource section (`same`). The same section indicates that all processes of a parallel job must run on the same type of host.

## Which sections apply

Depending on the command, one or more of these sections may apply. For example:

- ◆ `bsub` uses all sections
- ◆ `lshosts` only selects hosts, but does not order them
- ◆ `lslload` selects and orders hosts
- ◆ `lsqlace` uses the information in `select`, `order`, and `rusage` sections to select an appropriate host for a task
- ◆ `lslloadadj` uses the `rusage` section to determine how the load information should be adjusted on a host

## Syntax

```
select[selection_string] order[order_string] rusage[usage_string
[, usage_string][| usage_string ...] span[span_string]
same[same_string]
```

With the `bsub` and `bmod` commands, you can specify multiple `-R` `order`, `same`, `rusage`, and `select` sections. The `bmod` command does not support the use of the `||` operator.

The square brackets must be typed as shown.

The section names are `select`, `order`, `rusage`, `span`, and `same`. Sections that do not apply for a command are ignored.

If no section name is given, then the entire string is treated as a selection string. The `select` keyword may be omitted if the selection string is the first string in the resource requirement.

Each section has a different syntax.

By default, memory (`mem`) and swap (`swp`) limits in `select[]` and `rusage[]` sections are specified in MB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for these limits (MB, GB, TB, PB, or EB).

## How queue-level and job-level requirements are resolved

If job-level resource requirements are specified together with queue-level resource requirements:

## Selection String

- ◆ In a `select` string, a host must satisfy *both* queue-level and job-level requirements for the job to be dispatched.
- ◆ `order` and `span` sections defined at the queue level are ignored if different `order` and `span` requirements are specified at the job level. The default `order` string is `r15s:pg`.
- ◆ An `order` section defined at the queue level is ignored if different `order` requirements are specified at the job level. The default `order` string is `r15s:pg`.
- ◆ For usage strings, the `rusage` section defined for the job overrides the `rusage` section defined in the queue. The two `rusage` definitions are merged, with the job-level `rusage` taking precedence.

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements that exceed the requirements specified at the queue level.

## In application profiles

See Chapter 20, “[Working with Application Profiles](#)” for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

## Selection String

The selection string specifies the characteristics a host must have to match the resource requirement. It is a logical expression built from a set of resource names. The selection string is evaluated for each host; if the result is non-zero, then that host is selected.

## Syntax

The selection string can combine resource names with logical and arithmetic operators. Non-zero arithmetic values are treated as logical TRUE, and zero (0) as logical FALSE. Boolean resources (for example, `server` to denote LSF server hosts) have a value of one (1) if they are defined for a host, and zero (0) if they are not defined for the host.

The resource names `swap`, `idle`, `login`, and `cpu` are accepted as aliases for `swp`, `it`, `ls`, and `r1m` respectively.

For `ut`, specify the percentage CPU utilization as an integer between 0-100.

For the string resources `type` and `model`, the special value `any` selects any value and `local` selects the same value as that of the local host. For example, `type==local` selects hosts of the same type as the host submitting the job. If a job can run on any type of host, include `type==any` in the resource requirements.

If no `type` is specified, the default depends on the command. For `bsub`, `lspplace`, `lsrun`, and `lsgrun` the default is `type==local` unless a string or Boolean resource is specified, in which case it is `type==any`. For `lshosts`, `lsload`, `lsmon` and `lslogin` the default is `type==any`.

## Specifying multiple -R options

`bsub` accepts multiple `-R` options for the `select` section. You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "select[swp > 15]" -R "select[hpux]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

## Selecting shared string resources

You must use single quote characters (') around string-type shared resources. For example, use `lsload -s` to see the shared resources defined for the cluster:

```
lsload -s
RESOURCE VALUE LOCATION
os_version 4.2 pc36
os_version 4.0 pc34
os_version 4.1 devlinux4
cpu_type ia pc36
cpu_type ia pc34
cpu_type unknown devlinux4
```

Use a select string in `lsload -R` to specify the shared resources you want to view, enclosing the shared resource values in single quotes. For example:

```
lsload -R "select[os_version=='4.2' || cpu_type=='unknown']"
HOST_NAME status r15s r1m r15m ut pg ls it tmp swp mem
pc36 ok 0.0 0.2 0.1 1% 3.4 3 0 895M 517M 123M
devlinux4 ok 0.0 0.1 0.0 0% 2.8 4 0 6348M 504M 205M
```

## Operators

These operators can be used in selection strings. The operators are listed in order of decreasing precedence.

| Syntax                      | Meaning                                                  |
|-----------------------------|----------------------------------------------------------|
| <code>-a</code>             | Negative of a                                            |
| <code>!a</code>             | Logical not: 1 if a==0, 0 otherwise                      |
| <code>a * b</code>          | Multiply a and b                                         |
| <code>a / b</code>          | Divide a by b                                            |
| <code>a + b</code>          | Add a and b                                              |
| <code>a - b</code>          | Subtract b from a                                        |
| <code>a &gt; b</code>       | 1 if a is greater than b, 0 otherwise                    |
| <code>a &lt; b</code>       | 1 if a is less than b, 0 otherwise                       |
| <code>a &gt;= b</code>      | 1 if a is greater than or equal to b, 0 otherwise        |
| <code>a &lt;= b</code>      | 1 if a is less than or equal to b, 0 otherwise           |
| <code>a == b</code>         | 1 if a is equal to b, 0 otherwise                        |
| <code>a != b</code>         | 1 if a is not equal to b, 0 otherwise                    |
| <code>a &amp;&amp; b</code> | Logical AND: 1 if both a and b are non-zero, 0 otherwise |
| <code>a    b</code>         | Logical OR: 1 if either a or b is non-zero, 0 otherwise  |

## Examples

```
select[(swp > 50 && type == MIPS) || (swp > 35 && type == ALPHA)]
select[((2*r15s + 3*r1m + r15m) / 6 < 1.0) && !fs && (cpuf > 4.0)]
```

## Specifying shared resources with the keyword "defined"

A shared resource may be used in the resource requirement string of any LSF command. For example, when submitting an LSF job that requires a certain amount of shared scratch space, you might submit the job as follows:

```
bsub -R "avail_scratch > 200 && swap > 50" myjob
```

The above assumes that all hosts in the cluster have access to the shared scratch space. The job is only scheduled if the value of the "avail\_scratch" resource is more than 200 MB and goes to a host with at least 50 MB of available swap space.

It is possible for a system to be configured so that only some hosts within the LSF cluster have access to the scratch space. To exclude hosts that cannot access a shared resource, the `defined(resource_name)` function must be specified in the resource requirement string.

For example:

```
bsub -R "defined(avail_scratch) && avail_scratch > 100 && swap > 100" myjob
```

would exclude any hosts that cannot access the scratch resource. The LSF administrator configures which hosts do and do not have access to a particular shared resource.

## Specifying exclusive resources

An exclusive resource may be used in the resource requirement string of any placement or scheduling command, such as `bsub`, `lspplace`, `lsrun`, or `lsgrun`. An exclusive resource is a special resource that is assignable to a host. This host will not receive a job unless that job explicitly requests the host. For example, use the following command to submit a job requiring the exclusive resource `bigmem`:

```
bsub -R "bigmem" myjob
```

Jobs will not be submitted to the host with the `bigmem` resource unless the command uses the `-R` option to explicitly specify "bigmem".

To configure an exclusive resource, first define a static Boolean resource in `lsf.shared`. For example:

```
Begin Resource
...
bigmem Boolean () ()
End Resource
```

Assign the resource to a host in the Host section of `lsf.cluster.cluster_name`. Prefix the resource name with an exclamation mark (!) to indicate that the resource is exclusive to the host. For example:

```
Begin Host
HOSTNAME model type server r1m pg tmp RESOURCES RUNWINDOW
...
hostE ! ! 1 3.5 () () (linux !bigmem) ()
...
End Host
```

## Order String

The order string allows the selected hosts to be sorted according to the values of resources. The values of `r15s`, `r1m`, and `r15m` used for sorting are the normalized load indices returned by `lsload -N`.

The order string is used for host sorting and selection. The ordering begins with the rightmost index in the order string and proceeds from right to left. The hosts are sorted into order based on each load index, and if more hosts are available than were requested, the LIM drops the least desirable hosts according to that index. The remaining hosts are then sorted by the next index.

After the hosts are sorted by the leftmost index in the order string, the final phase of sorting orders the hosts according to their status, with hosts that are currently not available for load sharing (that is, not in the `ok` state) listed at the end.

Because the hosts are sorted again for each load index, only the host status and the leftmost index in the order string actually affect the order in which hosts are listed. The other indices are only used to drop undesirable hosts from the list.

When sorting is done on each index, the direction in which the hosts are sorted (increasing vs. decreasing values) is determined by the default order returned by `lsinfo` for that index. This direction is chosen such that after sorting, by default, the hosts are ordered from best to worst on that index.

### Syntax

```
[-]resource_name [:[-]resource_name]...
```

You can specify any built-in or external load index.

When an index name is preceded by a minus sign '`-`', the sorting order is reversed so that hosts are ordered from worst to best on that index.

### Specifying multiple `-R` options

`bsub` accepts multiple `-R` options for the order section. You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "order[r15m]" -R "order[ut]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

### Default

The default sorting order is `r15s:pg` (except for `lslogin(1):ls:r1m`).

### Example

```
swp:r1m:tmp:r15s
```

## Usage String

This string defines the expected resource usage of the job. It is used to specify resource reservations for jobs, or for mapping jobs on to hosts and adjusting the load when running interactive jobs.

## Usage String

By default, no resources are reserved.

## Batch jobs

The resource usage (`rusage`) section can be specified at the job level, with the queue configuration parameter `RES_REQ`, or the application profile parameter `RES_REQ`.

## Syntax

```
rusage[usage_string [, usage_string][| usage_string] ...]
```

where `usage_string` is:

```
load_index=value [:load_index=value]... [:duration=minutes[m] |
:duration=hoursh | :duration=secondss [:decay=0 | :decay=1]]
```

## Load index

Internal and external load indices are considered in the resource usage string. The resource value represents the initial reserved amount of the resource.

## Duration

The duration is the time period within which the specified resources should be reserved. Specify a duration equal to or greater than the ELIM updating interval.

- ◆ If the value is followed by the letter `s`, `m`, or `h`, the specified time is measured in seconds, minutes, or hours respectively.
- ◆ By default, duration is specified in minutes. For example, the following specify a duration of 1 hour:
  - ◆ `duration=60`
  - ◆ `duration=1h`
  - ◆ `duration=3600s`

---

**TIP:** Duration is not supported for static shared resources. If the shared resource is defined in an `lsb.resources Limit` section, then duration is not applied.

---

## Decay

The decay value indicates how the reserved amount should decrease over the duration.

- ◆ A value of 1 indicates that system should linearly decrease the amount reserved over the duration.
- ◆ A value of 0 causes the total amount to be reserved for the entire duration.

Values other than 0 or 1 are unsupported. If duration is not specified, decay value is ignored.

---

**TIP:** Decay is not supported for static shared resources. If the shared resource is defined in an `lsb.resources Limit` section, then decay is not applied.

---

## Default

If a resource or its value is not specified, the default is not to reserve that resource. If duration is not specified, the default is to reserve the total amount for the lifetime of the job. The default decay value is 0.

## Example

```
rusage[mem=50:duration=100:decay=1]
```

This example indicates that 50 MB memory should be reserved for the job. As the job runs, the amount reserved will decrease at approximately 0.5 MB per minute until the 100 minutes is up.



### How queue-level and job-level rusage sections are resolved

Job-level rusage overrides the queue-level specification:

- ◆ For internal load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `ls`, `it`, `tmp`, `swp`, and `mem`), the job-level value cannot be larger than the queue-level value.
- ◆ For external load indices (e.g., licenses), the job-level rusage can be larger than the queue-level requirements.
- ◆ For duration, the job-level value of internal and external load indices cannot be larger than the queue-level value.

### How queue-level and job-level rusage sections are merged

When both job-level and queue-level `rusage` sections are defined, the `rusage` section defined for the job overrides the `rusage` section defined in the queue. The two `rusage` definitions are merged, with the job-level `rusage` taking precedence. For example:

Given a `RES_REQ` definition in a queue:

```
RES_REQ = rusage[mem=200:lic=1] ...
```

and job submission:

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is

```
rusage[mem=100:lic=1]
```

where `mem=100` specified by the job overrides `mem=200` specified by the queue. However, `lic=1` from queue is kept, since job does not specify it.

For the following queue-level `RES_REQ` (decay and duration defined):

```
RES_REQ = rusage[mem=200:duration=20:decay=1] ...
```

and job submission (no decay or duration):

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is:

```
rusage[mem=100:duration=20:decay=1]
```

Queue-level duration and decay are merged with the job-level specification, and `mem=100` for the job overrides `mem=200` specified by the queue. However, `duration=20` and `decay=1` from queue are kept, since job does not specify them.

### rusage in application profiles

See Chapter 20, “[Working with Application Profiles](#)” for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

## Specifying multiple -R options

`bsub` accepts multiple `-R` options for the `rusage` section. You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "rusage[mem=100]" -R "rusage[tmp=50:duration=60]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

Comma-separated multiple resource requirements within one `rusage` string is supported. For example:

```
bsub -R "rusage[mem=20, license=1:duration=2]"
-R "rusage[app_lic_v201=1||app_lic_v15=1]" myjob
```

A given load index cannot appear more than once in the resource usage string.

## Examples

- ◆ The following job requests 20 MB memory for the duration of the job, and 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[mem=20, license=1:duration=2]" myjob
```

- ◆ A queue with the same resource requirements could specify:

```
RES_REQ = rusage[mem=20, license=1:duration=2]
```

- ◆ The following job requests 20 MB of memory and 50 MB of swap space for 1 hour, and 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[mem=20:swp=50:duration=1h, license=1:duration=2]" myjob
```

- ◆ The following job requests 50 MB of swap space, linearly decreasing the amount reserved over a duration of 2 hours, and requests 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[swp=20:duration=2h:decay=1, license=1:duration=2]" myjob
```

- ◆ The following job requests two resources with same duration but different decay:

```
bsub -R "rusage[mem=20:duration=30:decay=1, lic=1:duration=30] myjob
```

## Specifying alternative usage strings

If you use more than one version of an application, you can specify the version you prefer to use together with a legacy version you can use if the preferred version is not available. Use the OR (|) expression to separate the different usage strings that define your alternative resources.

Job-level resource requirement specifications that use the || operator are merged with other `rusage` requirements defined at the application and queue levels.

## Job-level examples

The following examples assume that you are running an application version 1.5 as a resource called `app_lic_v15` and the same application version 2.0.1 as a resource called `app_lic_v201`. The license key for version 2.0.1 is backward compatible with version 1.5, but the license key for version 1.5 will not work with 2.0.1

- ◆ If you can only run your job using version 2.0.1 of the application, submit the job without specifying an alternate resource. To submit a job that will only use `app_lic_v201`:

```
bsub -R "rusage[app_lic_v201=1]" myjob
```

- ◆ If you can run your job using either version of the application, try to reserve version 2.0.1 of the application. If it is not available, you can use version 1.5. To submit a job that will try `app_lic_v201` before trying `app_lic_v15`:

```
bsub -R "rusage[app_lic_v201=1||app_lic_v15=1]" myjob
```

- ◆ If different versions of an application require different system resources, you can specify other resources in your `rusage` strings. To submit a job that will use 20 MB of memory for `app_lic_v201` or 20 MB of memory and 50 MB of swap space for `app_lic_v15`:

```
bsub -R "rusage[mem=20:app_lic_v201=1||mem=20:swp=50:app_lic_v15=1]" myjob
```

## How LSF merges rusage strings that contain the || operator

The following examples show how LSF merges job-level and queue-level `rusage` strings that contain the `||` operator.

| Queue level RES_REQ=rusage...       | Job level bsub -R "rusage ..."       | Resulting rusage string                                                                                       |
|-------------------------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>[mem=200:duration=180]</code> | <code>[w1=1    w2=1    w3=1]"</code> | <code>[w1=1: mem=200:duration=180   <br/>w2=1: mem=200:duration=180   <br/>w3=1: mem=200:duration=180]</code> |
| <code>[w1=1    w2=1    w3=1]</code> | <code>[mem=200:duration=180]"</code> | <code>[mem=200:duration=180:w1=1   <br/>mem=200:duration=180:w2=1   <br/>mem=200:duration=180:w3=1]</code>    |

## Non-batch environments

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, such as through `lsrun`, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]: ... :res[=value]
```

`res` is one of the resources whose value is returned by the `lsload` command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

## Span String

A `span` string specifies the locality of a parallel job. If `span` is omitted, LSF allocates the required processors for the job from the available set of processors.

### Syntax

The `span` string supports the following syntax:

- span[hosts=1]** Indicates that all the processors allocated to this job must be on the same host.
- span[ptile=value]** Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:
  - ◆ Default `ptile` value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:
 

```
span[ptile=4]
```

## Same String

- ◆ Predefined `ptile` value, specified by '!'. The following example uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value:

```
span[ptile='!']
```

---

**TIP:** If the host or host type/model does not define MXJ, the default predefined `ptile` value is 1.

---

**RESTRICTION:** Under bash 3.0, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined `ptile` value (`ptile='!'`), use the `+H` option to disable '!' style history substitution in bash (`sh +H`).

---

- ◆ Predefined `ptile` value with optional multiple `ptile` values, per host type or host model:

- ❖ For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type `HP` or `SGI`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types:

```
span[ptile='!',HP:8,SGI:8,LINUX:2] same[type]
```

- ❖ For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models:

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

**span[hosts=-1]** Disables `span` setting in the queue. LSF allocates the required processors for the job from the available set of processors.

See [Controlling Processor Allocation Across Hosts](#) on page 474 for more information about specifying `span` strings.

## Same String

---

**TIP:** You must have the parallel batch job scheduler plugin installed in order to use the `same` string.

---

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts, some processes from a parallel job may for example, run on Solaris and some on SGI IRIX. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

The *same* string specifies that all processes of a parallel job must run on hosts with the same resource.

You can specify the *same* string:

- ◆ At the job level in the resource requirement string of:
  - ❖ `bsub`
  - ❖ `bmod`

- ◆ At the queue-level in `lsb.queues` in the `RES_REQ` parameter.

When both queue-level and job-level `same` sections are defined, LSF combines both requirements to allocate processors.

## Syntax

```
resource_name[:resource_name]...
```

You can specify any static resource.

For example, if you specify `resource1:resource2`, if hosts always have both resources, the string is interpreted as allocate processors only on hosts that have the same value for `resource1` and the same value for `resource2`.

If hosts do not always have both resources, it is interpreted as allocate processors either on hosts that have the same value for `resource1`, or on hosts that have the same value for `resource2`, or on hosts that have the same value for both `resource1` and `resource2`.

## Specifying multiple -R options

`bsub` accepts multiple `-R` options for the same section. You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "same[type]" -R "same[model]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

## Examples

```
bsub -n 4 -R"select[type==SGI6 || type==SOL7] same[type]" myjob
```

Run all parallel processes on the same host type. Allocate 4 processors on the same host type—either SGI IRIX, or Solaris 7, but not both.

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Run all parallel processes on the same host type and model. Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

### Same string in application profiles

See Chapter 20, “[Working with Application Profiles](#)” for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Same String

## Fairshare Scheduling

To configure any kind of fairshare scheduling, you should understand the following concepts:

- ◆ User share assignments
- ◆ Dynamic share priority
- ◆ Job dispatch order

You can configure fairshare at either host level or queue level. If you require more control, you can implement hierarchical fairshare. You can also set some additional restrictions when you submit a job.

To get ideas about how to use fairshare scheduling to do different things, see [Ways to Configure Fairshare](#) on page 324.

### Contents

- ◆ Basic Concepts
  - ❖ [Understanding Fairshare Scheduling](#) on page 296
  - ❖ [User Share Assignments](#) on page 297
  - ❖ [Dynamic User Priority](#) on page 298
  - ❖ [How Fairshare Affects Job Dispatch Order](#) on page 300
- ◆ User-based Fairshare
  - ❖ [Host Partition User-based Fairshare](#) on page 301
  - ❖ [Queue-level User-based Fairshare](#) on page 302
  - ❖ [Cross-queue User-based Fairshare](#) on page 302
  - ❖ [Hierarchical User-based Fairshare](#) on page 306
- ◆ Queue-based Fairshare
  - ❖ [Queue-based Fairshare](#) on page 309
  - ❖ [Configuring Slot Allocation per Queue](#) on page 311
  - ❖ [View Queue-based Fairshare Allocations](#) on page 313
- ◆ Advanced Topics

- ❖ [Using Historical and Committed Run Time](#) on page 319
- ❖ [Users Affected by Multiple Fairshare Policies](#) on page 323
- ❖ [Ways to Configure Fairshare](#) on page 324

## Understanding Fairshare Scheduling

By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.

Fairshare scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

If your cluster has many users competing for limited resources, the FCFS policy might not be enough. For example, one user could submit many long jobs at once and monopolize the cluster's resources for a long time, while other users submit urgent jobs that must wait in queues until all the first user's jobs are all done. To prevent this, use fairshare scheduling to control how resources should be shared by competing users.

Fairshare is not necessarily equal share: you can assign a higher priority to the most important users. If there are two users competing for resources, you can:

- ◆ Give all the resources to the most important user
- ◆ Share the resources so the most important user gets the most resources
- ◆ Share the resources so that all users have equal importance

### Queue-level vs. host partition fairshare

You can configure fairshare at either the queue level or the host level. However, these types of fairshare scheduling are mutually exclusive. You cannot configure queue-level fairshare and host partition fairshare in the same cluster.

If you want a user's priority in one queue to depend on their activity in another queue, you must use cross-queue fairshare or host-level fairshare.

### Fairshare policies

A fairshare policy defines the order in which LSF attempts to place jobs that are in a queue or a host partition. You can have multiple fairshare policies in a cluster, one for every different queue or host partition. You can also configure some queues or host partitions with fairshare scheduling, and leave the rest using FCFS scheduling.

### How fairshare scheduling works

Each fairshare policy assigns a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares. Users who have no shares cannot run jobs in the queue or host partition.

A user's dynamic priority depends on their share assignment, the dynamic priority formula, and the resources their jobs have already consumed.



The order of jobs in the queue is secondary. The most important thing is the dynamic priority of the user who submitted the job. When fairshare scheduling is used, LSF tries to place the first job in the queue that belongs to the user with the highest dynamic priority.

## User Share Assignments

Both queue-level and host partition fairshare use the following syntax to define how shares are assigned to users or user groups.

### Syntax

`[user, number_shares]`

Enclose each user share assignment in square brackets, as shown. Separate multiple share assignments with a space between each set of square brackets.

**user** Specify users of the queue or host partition. You can assign the shares:

- ◆ to a single user (specify *user\_name*)
- ◆ to users in a group, individually (specify *group\_name@*) or collectively (specify *group\_name*)
- ◆ to users not included in any other share assignment, individually (specify the keyword `default`) or collectively (specify the keyword `others`)

By default, when resources are assigned collectively to a group, the group members compete for the resources according to FCFS scheduling. You can use hierarchical fairshare to further divide the shares among the group members.

When resources are assigned to members of a group individually, the share assignment is recursive. Members of the group and of all subgroups always compete for the resources according to FCFS scheduling, regardless of hierarchical fairshare policies.

**number\_shares** Specify a positive integer representing the number of shares of cluster resources assigned to the user.

The number of shares assigned to each user is only meaningful when you compare it to the shares assigned to other users, or to the total number of shares. The total number of shares is just the sum of all the shares assigned in each share assignment.

### Examples

```
[User1, 1] [GroupB, 1]
```

Assigns 2 shares: 1 to User1, and 1 to be shared by the users in GroupB. Each user in GroupB has equal importance. User1 is as important as all the users in GroupB put together. In this example, it does not matter if the number of shares is 1, 6 or 600. As long as User1 and GroupB are both assigned the same number of shares, the relationship stays the same.

```
[User1, 10] [GroupB@, 1]
```

If GroupB contains 10 users, assigns 20 shares in total: 10 to User1, and 1 to each user in GroupB. Each user in GroupB has equal importance. User1 is ten times as important as any user in GroupB.

```
[User1, 10] [User2, 9] [others, 8]
```

Assigns 27 shares: 10 to User1, 9 to User2, and 8 to the remaining users, as a group. User1 is slightly more important than User2. Each of the remaining users has equal importance.

- ◆ If there are 3 users in total, the single remaining user has all 8 shares, and is almost as important as User1 and User2.
- ◆ If there are 12 users in total, then 10 users compete for those 8 shares, and each of them is significantly less important than User1 and User2.

```
[User1, 10] [User2, 6] [default, 4]
```

The relative percentage of shares held by a user will change, depending on the number of users who are granted shares by default.

- ◆ If there are 3 users in total, assigns 20 shares: 10 to User1, 6 to User2, and 4 to the remaining user. User1 has half of the available resources (10 shares out of 20).
- ◆ If there are 12 users in total, assigns 56 shares: 10 to User1, 6 to User2, and 4 to each of the remaining 10 users. User1 has about a fifth of the available resources (10 shares out of 56).

## Dynamic User Priority

LSF calculates a *dynamic user priority* for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

### How LSF calculates dynamic priority

By default, LSF calculates the dynamic priority for each user based on:

- ◆ The number of shares assigned to the user
- ◆ The resources used by jobs belonging to the user:
  - ❖ Number of job slots reserved and in use
  - ❖ Run time of running jobs
  - ❖ Cumulative actual CPU time (not normalized), adjusted so that recently used CPU time is weighted more heavily than CPU time used in the distant past

If you enable additional functionality, the formula can also involve additional resources used by jobs belonging to the user:

- ◆ Historical run time of finished jobs
- ◆ Committed run time, specified at job submission with the `-W` option of `bsub`, or in the queue with the `RUNLIMIT` parameter in `lsb.queues`

### How LSF measures fairshare resource usage

LSF measures resource usage differently, depending on the type of fairshare:

- ◆ For user-based fairshare:

- ❖ For queue-level fairshare, LSF measures the resource consumption of all the user's jobs in the queue. This means a user's dynamic priority can be different in every queue.
- ❖ For host partition fairshare, LSF measures resource consumption for all the user's jobs that run on hosts in the host partition. This means a user's dynamic priority is the same in every queue that uses hosts in the same partition.
- ◆ For queue-based fairshare, LSF measures the resource consumption of all jobs in each queue.

## Default dynamic priority formula

By default, LSF calculates dynamic priority according to the following formula:

$$\text{dynamic priority} = \text{number\_shares} / (\text{cpu\_time} * \text{CPU\_TIME\_FACTOR} + \text{run\_time} * \text{RUN\_TIME\_FACTOR} + (1 + \text{job\_slots}) * \text{RUN\_JOB\_FACTOR})$$

The maximum value of dynamic user priority is 100 times the number of user shares (if the denominator in the calculation is less than 0.01, LSF rounds up to 0.01).

For *cpu\_time*, *run\_time*, and *job\_slots*, LSF uses the total resource consumption of all the jobs in the queue or host partition that belong to the user or group.

*number\_shares* The number of shares assigned to the user.

*cpu\_time* The cumulative CPU time used by the user (measured in hours). LSF calculates the cumulative CPU time using the actual (not normalized) CPU time and a decay factor such that 1 hour of recently-used CPU time decays to 0.1 hours after an interval of time specified by HIST\_HOURS in `lsb.params` (5 hours by default).

*run\_time* The total run time of running jobs (measured in hours).

*job\_slots* The number of job slots reserved and in use.

## Configuring the default dynamic priority

You can give additional weight to the various factors in the priority calculation by setting the following parameters in `lsb.params`.

- ◆ CPU\_TIME\_FACTOR
- ◆ RUN\_TIME\_FACTOR
- ◆ RUN\_JOB\_FACTOR
- ◆ HIST\_HOURS

If you modify the parameters used in the dynamic priority formula, it affects every fairshare policy in the cluster.

**CPU\_TIME\_FACTOR** The CPU time weighting factor.  
*Default: 0.7*

**RUN\_TIME\_FACTOR** The run time weighting factor.  
*Default: 0.7*

**RUN\_JOB\_FACTOR** The job slots weighting factor.

*Default: 3*

**HIST\_HOURS** Interval for collecting resource consumption history

*Default: 5*

## How Fairshare Affects Job Dispatch Order

Within a queue, jobs are dispatched according to the queue's scheduling policy.

- ◆ For FCFS queues, the dispatch order depends on the order of jobs in the queue (which depends on job priority and submission time, and can also be modified by the job owner).
- ◆ For fairshare queues, the dispatch order depends on dynamic share priority, then order of jobs in the queue (which is not necessarily the order in which they are submitted to the queue).

A user's priority gets higher when they use less than their fair share of the cluster's resources. When a user has the highest priority, LSF considers one of their jobs first, even if other users are ahead of them in the queue.

If there are only one user's jobs pending, and you do not use hierarchical fairshare, then there is no resource contention between users, so the fairshare policies have no effect and jobs are dispatched as usual.

### Job dispatch order among queues of equivalent priority

The order of dispatch depends on the order of the queues in the queue configuration file. The first queue in the list is the first to be scheduled.

Jobs in a fairshare queue are always considered as a group, so the scheduler attempts to place all jobs in the queue before beginning to schedule the next queue.

Jobs in an FCFS queue are always scheduled along with jobs from other FCFS queues of the same priority (as if all the jobs belonged to the same queue).

### Example

In a cluster, queues A, B, and C are configured in that order and have equal queue priority.

Jobs with equal job priority are submitted to each queue in this order: C B A B A.

- ◆ If all queues are FCFS queues, order of dispatch is C B A B A (queue A is first; queues B and C are the same priority as A; all jobs are scheduled in FCFS order).
- ◆ If all queues are fairshare queues, order of dispatch is AA BB C (queue A is first; all jobs in the queue are scheduled; then queue B, then C).
- ◆ If A and C are fairshare, and B is FCFS, order of dispatch is AA B B C (queue A jobs are scheduled according to user priority; then queue B jobs are scheduled in FCFS order; then queue C jobs are scheduled according to user priority)
- ◆ If A and C are FCFS, and B is fairshare, order of dispatch is C A A BB (queue A is first; queue A and C jobs are scheduled in FCFS order, then queue B jobs are scheduled according to user priority)

- ◆ If any of these queues uses cross-queue fairshare, the other queues must also use cross-queue fairshare and belong to the same set, or they cannot have the same queue priority. For more information, see [Cross-queue User-based Fairshare](#) on page 302.

## Host Partition User-based Fairshare

User-based fairshare policies configured at the host level handle resource contention across multiple queues.

You can define a different fairshare policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.

To run a job on a host that has fairshare, users must have a share assignment (`USER_SHARES` in the `HostPartition` section of `lsb.hosts`). Even cluster administrators cannot submit jobs to a fairshare host if they do not have a share assignment.

### View host partition information

- 1 Use `bhpart` to view the following information:

- ◆ Host partitions configured in your cluster
- ◆ Number of shares (for each user or group in a host partition)
- ◆ Dynamic share priority (for each user or group in a host partition)
- ◆ Number of started jobs
- ◆ Number of reserved jobs
- ◆ CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- ◆ Run time, in seconds (historical and actual run time for all members of the group, recursively)

```
% bhpart Partition1
HOST_PARTITION_NAME: Partition1
HOSTS: hostA hostB hostC
SHARE_INFO_FOR: Partition1/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
group1 100 5.440 5 0 200.0 1324
```

### Configure host partition fairshare scheduling

- 1 To configure host partition fairshare, define a host partition in `lsb.hosts`.

Use the following format.

```
Begin HostPartition
HPART_NAME = Partition1
HOSTS = hostA hostB ~hostC
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

- ◆ A host cannot belong to multiple partitions.

- ◆ Optional: Use the reserved host name `all` to configure a single partition that applies to all hosts in a cluster.
- ◆ Optional: Use the not operator (`~`) to exclude hosts or host groups from the list of hosts in the host partition.
- ◆ Hosts in a host partition cannot participate in queue-based fairshare. Hosts that are not included in any host partition are controlled by FCFS scheduling policy instead of fairshare scheduling policy.

## Queue-level User-based Fairshare

User-based fairshare policies configured at the queue level handle resource contention among users in the same queue. You can define a different fairshare policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.

To submit jobs to a fairshare queue, users must be allowed to use the queue (`USERS` in `lsb.queues`) and must have a share assignment (`FAIRSHARE` in `lsb.queues`). Even cluster and queue administrators cannot submit jobs to a fairshare queue if they do not have a share assignment.

### View queue-level fairshare information

- 1 To find out if a queue is a fairshare queue, run `bqueues -l`. If you see "USER\_SHARES" in the output, then a fairshare policy is configured for the queue.

### Configure queue-level fairshare

- 1 To configure a fairshare queue, define `FAIRSHARE` in `lsb.queues` and specify a share assignment for all users of the queue:

```
FAIRSHARE = USER_SHARES[[user, number_shares]...]
```

  - ◆ You must specify at least one user share assignment.
  - ◆ Enclose the list in square brackets, as shown.
  - ◆ Enclose each user share assignment in square brackets, as shown.

## Cross-queue User-based Fairshare

User-based fairshare policies configured at the queue level handle resource contention across multiple queues.

### Applying the same fairshare policy to several queues

With cross-queue fairshare, the same user-based fairshare policy can apply to several queues at the same time. You define the fairshare policy in a *master queue* and list *slave queues* to which the same fairshare policy applies; slave queues inherit the same fairshare policy as your master queue. For job scheduling purposes, this is equivalent to having one queue with one fairshare tree.

In this way, if a user submits jobs to different queues, user priority is calculated by taking into account all the jobs the user has submitted across the defined queues.

To submit jobs to a fairshare queue, users must be allowed to use the queue (USERS in `lsb.queues`) and must have a share assignment (FAIRSHARE in `lsb.queues`). Even cluster and queue administrators cannot submit jobs to a fairshare queue if they do not have a share assignment.

## User and queue priority

By default, a user has the same priority across the master and slave queues. If the same user submits several jobs to these queues, user priority is calculated by taking into account all the jobs the user has submitted across the master-slave set.

If DISPATCH\_ORDER=QUEUE is set in the master queue, jobs are dispatched according to queue priorities first, then user priority. This avoids having users with higher fairshare priority getting jobs dispatched from low-priority queues.

Jobs from users with lower fairshare priorities who have pending jobs in higher priority queues are dispatched before jobs in lower priority queues. Jobs in queues having the same priority are dispatched according to user priority.

Queues that are not part of the ordered cross-queue fairshare can have any priority. Their priority can fall within the priority range of cross-queue fairshare queues and they can be inserted between two queues using the same fairshare tree.

## View cross-queue fairshare information

- 1 Run `bqueues -l` to know if a queue is part of cross-queue fairshare.

The FAIRSHARE\_QUEUES parameter indicates cross-queue fairshare. The first queue listed in the FAIRSHARE\_QUEUES parameter is the master queue—the queue in which fairshare is configured; all other queues listed inherit the fairshare policy from the master queue.

All queues that participate in the same cross-queue fairshare display the same fairshare information (SCHEDULING POLICIES, FAIRSHARE\_QUEUES, USER\_SHARES, SHARE\_INFO\_FOR) when `bqueues -l` is used. Fairshare information applies to all the jobs running in all the queues in the master-slave set.

`bqueues -l` also displays DISPATCH\_ORDER in the master queue if it is defined.

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
normal 30 Open:Active - - - - - 1 1 0 0
short 40 Open:Active - 4 2 - - 1 0 1 0
license 50 Open:Active 10 1 1 - - 1 0 1 0

bqueues -l normal
QUEUE: normal
-- For normal low priority jobs, running only if hosts are lightly loaded. This is
the default queue.

PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
30 20 Open:Inact_Win - - - - - 1 1 0 0 0 0
```

## Cross-queue User-based Fairshare

```
SCHEDULING PARAMETERS
r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

SCHEDULING POLICIES: FAIRSHARE
FAIRSHARE_QUEUES: normal short license
USER_SHARES: [user1, 100] [default, 1]
SHARE_INFO_FOR: normal/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
user1 100 9.645 2 0 0.2 7034
USERS: all users
HOSTS: all
...
bqueues -l short
QUEUE: short
PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
40 20 Open:Inact_Win - 4 2 - 1 0 1 0 0 0

SCHEDULING PARAMETERS
r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

SCHEDULING POLICIES: FAIRSHARE
FAIRSHARE_QUEUES: normal short license
USER_SHARES: [user1, 100] [default, 1]
SHARE_INFO_FOR: short/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
user1 100 9.645 2 0 0.2 7034
USERS: all users
HOSTS: all
...
```

---

## Configuring cross-queue fairshare

### Considerations

- ◆ FAIRSHARE must be defined in the master queue. If it is also defined in the queues listed in FAIRSHARE\_QUEUES, it will be ignored.
- ◆ Cross-queue fairshare can be defined more than once within `lsb.queues`. You can define several sets of master-slave queues. However, a queue cannot belong to more than one master-slave set. For example, you can define:



- ❖ In master queue normal: FAIRSHARE\_QUEUES=short license
- ❖ In master queue priority: FAIRSHARE\_QUEUES= night owners

You cannot, however, define night, owners, or priority as slaves in the normal queue; or normal, short and license as slaves in the priority queue; or short, license, night, owners as master queues of their own.

- ◆ Cross-queue fairshare cannot be used with host partition fairshare. It is part of queue-level fairshare.

## Configure cross-queue fairshare

- 1 Decide to which queues in your cluster cross-queue fairshare will apply.

For example, in your cluster you may have the queues normal, priority, short, and license and you want cross-queue fairshare to apply only to normal, license, and short.

- 2 Define fairshare policies in your master queue.

In the queue you want to be the master, for example normal, define the following in `lsb.queues`:

- ◆ FAIRSHARE and specify a share assignment for all users of the queue.
- ◆ FAIRSHARE\_QUEUES and list slave queues to which the defined fairshare policy will also apply
- ◆ PRIORITY to indicate the priority of the queue.

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY = 30
NICE = 20
FAIRSHARE = USER_SHARES[[user1,100] [default,1]]
FAIRSHARE_QUEUES = queue2 queue3
DESCRIPTION = For normal low priority jobs, running only if hosts
 are lightly loaded.
End Queue
```

- 3 In all the slave queues listed in FAIRSHARE\_QUEUES, define all queue values as desired.

For example:

```
Begin Queue
QUEUE_NAME = queue2
PRIORITY = 40
NICE = 20
UJOB_LIMIT = 4
PJOB_LIMIT = 2
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY = 50
NICE = 10
PREEMPTION = PREEMPTIVE
QJOB_LIMIT = 10
UJOB_LIMIT = 1
PJOB_LIMIT = 1
End Queue
```

## Controlling job dispatch order in cross-queue fairshare

**DISPATCH\_ORDER parameter (lsb.queues)** Use DISPATCH\_ORDER=QUEUE in the master queue to define an *ordered* cross-queue fairshare set. DISPATCH\_ORDER indicates that jobs are dispatched according to the order of queue priorities, not user fairshare priority.

**Priority range in cross-queue fairshare** By default, the range of priority defined for queues in cross-queue fairshare cannot be used with any other queues. The priority of queues that are not part of the cross-queue fairshare cannot fall between the priority range of cross-queue fairshare queues.

For example, you have 4 queues: `queue1`, `queue2`, `queue3`, and `queue4`. You configure cross-queue fairshare for `queue1`, `queue2`, and `queue3`, and assign priorities of 30, 40, 50 respectively. The priority of `queue4` (which is not part of the cross-queue fairshare) cannot fall between 30 and 50, but it can be any number up to 29 or higher than 50. It does not matter if `queue4` is a fairshare queue or FCFS queue.

If DISPATCH\_ORDER=QUEUE is set in the master queue, queues that are not part of the ordered cross-queue fairshare can have any priority. Their priority can fall within the priority range of cross-queue fairshare queues and they can be inserted between two queues using the same fairshare tree. In the example above, `queue4` can have any priority, including a priority falling between the priority range of the cross-queue fairshare queues (30-50).

- Jobs from equal priority queues**
- ◆ If two or more *non-fairshare* queues have the same priority, their jobs are dispatched first-come, first-served based on submission time or job ID as if they come from the same queue.
  - ◆ If two or more *fairshare* queues have the same priority, jobs are dispatched in the order the queues are listed in `lsb.queues`.

## Hierarchical User-based Fairshare

For both queue and host partitions, hierarchical user-based fairshare lets you allocate resources to users in a hierarchical manner.

By default, when shares are assigned to a group, group members compete for resources according to FCFS policy. If you use hierarchical fairshare, you control the way shares that are assigned collectively are divided among group members.

If groups have subgroups, you can configure additional levels of share assignments, resulting in a multi-level share tree that becomes part of the fairshare policy.

### How hierarchical fairshare affects dynamic share priority

When you use hierarchical fairshare, the dynamic share priority formula does not change, but LSF measures the resource consumption for all levels of the share tree. To calculate the dynamic priority of a group, LSF uses the resource consumption of all the jobs in the queue or host partition that belong to users in the group and all its subgroups, recursively.

## How hierarchical fairshare affects job dispatch order

LSF uses the dynamic share priority of a user or group to find out which user's job to run next. If you use hierarchical fairshare, LSF works through the share tree from the top level down, and compares the dynamic priority of users and groups at each level, until the user with the highest dynamic priority is a single user, or a group that has no subgroups.

## View hierarchical share information for a group

- 1 Use `bugroup -l` to find out if you belong to a group, and what the share distribution is.

```
bugroup -l
GROUP_NAME: group1
USERS: group2/ group3/
SHARES: [group2,20] [group3,10]
GROUP_NAME: group2
USERS: user1 user2 user3
SHARES: [others,10] [user3,4]
GROUP_NAME: group3
USERS: all
SHARES: [user2,10] [default,5]
```

This command displays all the share trees that are configured, even if they are not used in any fairshare policy.

## View hierarchical share information for a host partition

By default, `bhpart` displays only the top level share accounts associated with the partition.

- 1 Use `bhpart -r` to display the group information recursively.

The output lists all the groups in the share tree, starting from the top level, and displays the following information:

- ◆ Number of shares
- ◆ Dynamic share priority (LSF compares dynamic priorities of users who belong to same group, at the same level)
- ◆ Number of started jobs
- ◆ Number of reserved jobs
- ◆ CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- ◆ Run time, in seconds (historical and actual run time for all members of the group, recursively)

```
bhpart -r Partition1
HOST_PARTITION_NAME: Partition1
HOSTS: HostA
```

## Hierarchical User-based Fairshare

```
SHARE_INFO_FOR: Partition1/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
group1 40 1.867 5 0 48.4 17618
group2 20 0.775 6 0 607.7 24664
SHARE_INFO_FOR: Partition1/group2/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
user1 8 1.144 1 0 9.6 5108
user2 2 0.667 0 0 0.0 0
others 1 0.046 5 0 598.1 19556
```

---

## Configuring hierarchical fairshare

To define a hierarchical fairshare policy, configure the top-level share assignment in `lsb.queues` or `lsb.hosts`, as usual. Then, for any group of users affected by the fairshare policy, configure a share tree in the `UserGroup` section of `lsb.users`. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.

If shares are assigned to members of any group individually, using `@`, there can be no further hierarchical fairshare within that group. The shares are assigned recursively to all members of all subgroups, regardless of further share distributions defined in `lsb.users`. The group members and members of all subgroups compete for resources according to FCFS policy.

You can choose to define a hierarchical share tree for some groups but not others. If you do not define a share tree for any group or subgroup, members compete for resources according to FCFS policy.

## Configure a share tree

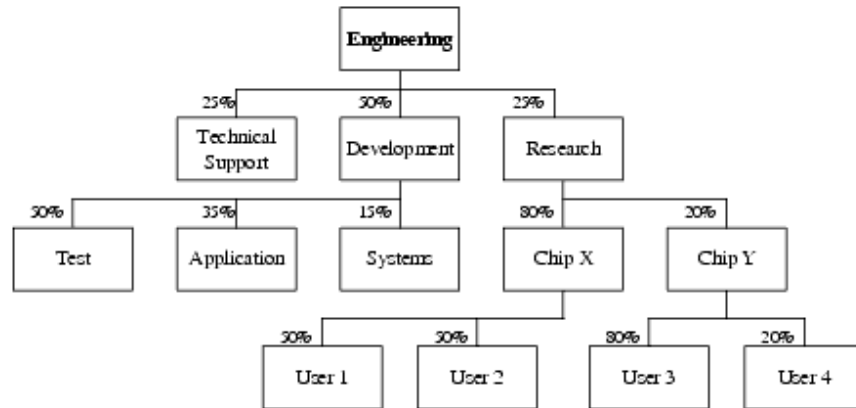
- 1 Group membership is already defined in the `UserGroup` section of `lsb.users`. To configure a share tree, use the `USER_SHARES` column to describe how the shares are distributed in a hierarchical manner. Use the following format.

```
Begin UserGroup
GROUP_NAME GROUP_MEMBER USER_SHARES
GroupB (User1 User2) ()
GroupC (User3 User4) ([User3, 3] [User4, 4])
GroupA (GroupB GroupC User5) ([User5, 1] [default, 10])
End UserGroup
```

- ◆ User groups must be defined before they can be used (in the `GROUP_MEMBER` column) to define other groups.
- ◆ Enclose the share assignment list in parentheses, as shown, even if you do not specify any user share assignments.

---

An Engineering queue or host partition organizes users hierarchically, and divides the shares as shown. It does not matter what the actual number of shares assigned at each level is.



The Development group gets the largest share (50%) of the resources in the event of contention. Shares assigned to the Development group can be further divided among the Systems, Application, and Test groups, which receive 15%, 35%, and 50%, respectively. At the lowest level, individual users compete for these shares as usual.

One way to measure a user's importance is to multiply their percentage of the resources at every level of the share tree. For example, *User1* is entitled to 10% of the available resources ( $.50 \times .80 \times .25 = .10$ ) and *User3* is entitled to 4% ( $.80 \times .20 \times .25 = .04$ ). However, if Research has the highest dynamic share priority among the 3 groups at the top level, and ChipY has a higher dynamic priority than ChipX, the next comparison is between *User3* and *User4*, so the importance of *User1* is not relevant. The dynamic priority of *User1* is not even calculated at this point.

## Queue-based Fairshare

When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.

*Queue-based fairshare* allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a *soft job slot limit* on a queue. This allows you to organize the priorities of your work and tune the number of jobs dispatched from a queue so that no single queue monopolizes cluster resources, leaving other queues waiting to dispatch jobs.

You can balance the distribution of job slots among queues by configuring a ratio of jobs waiting to be dispatched from each queue. LSF then attempts to dispatch a certain percentage of jobs from each queue, and does not attempt to drain the highest priority queue entirely first.

When queues compete, the allocated slots per queue are kept within the limits of the configured share. If only one queue in the pool has jobs, that queue can use all the available resources and can span its usage across all hosts it could potentially run jobs on.

## Managing pools of queues

You can configure your queues into a *pool*, which is a named group of queues using the same set of hosts. A pool is entitled to a slice of the available job slots. You can configure as many pools as you need, but each pool must use the same set of hosts. There can be queues in the cluster that do not belong to any pool yet share some hosts used by a pool.

## How LSF allocates slots for a pool of queues

During job scheduling, LSF orders the queues within each pool based on the shares the queues are entitled to. The number of running jobs (or job slots in use) is maintained at the percentage level specified for the queue. When a queue has no pending jobs, leftover slots are redistributed to other queues in the pool with jobs pending.

The total number of slots in each pool is constant; it is equal to the number of slots in use plus the number of free slots to the maximum job slot limit configured either in `lsb.hosts (MXJ)` or in `lsb.resources` for a host or host group. The accumulation of slots in use by the queue is used in ordering the queues for dispatch.

Job limits and host limits are enforced by the scheduler. For example, if LSF determines that a queue is eligible to run 50 jobs, but the queue has a job limit of 40 jobs, no more than 40 jobs will run. The remaining 10 job slots are redistributed among other queues belonging to the same pool, or make them available to other queues that are configured to use them.

### Accumulated slots in use

As queues run the jobs allocated to them, LSF accumulates the slots each queue has used and decays this value over time, so that each queue is not allocated more slots than it deserves, and other queues in the pool have a chance to run their share of jobs.

## Interaction with other scheduling policies

- ◆ Queues participating in a queue-based fairshare pool cannot be preemptive or preemptable.
- ◆ You should not configure slot reservation (`SLOT_RESERVE`) in queues that use queue-based fairshare.
- ◆ Cross-queue user-based fairshare (`FAIRSHARE_QUEUES`) can undo the dispatching decisions of queue-based fairshare. Cross-queue user-based fairshare queues should not be part of a queue-based fairshare pool.

## Examples

**Three queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots to be allocated:**

- ◆ `queue1` shares 50% of slots to be allocated =  $2 * 6 * 0.5 = 6$  slots
- ◆ `queue2` shares 30% of slots to be allocated =  $2 * 6 * 0.3 = 3.6 \rightarrow 4$  slots
- ◆ `queue3` shares 20% of slots to be allocated =  $2 * 6 * 0.2 = 2.4 \rightarrow 3$  slots; however, since the total cannot be more than 12, `queue3` is actually allocated only 2 slots.

**Four queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots; queue4 does not belong to any pool.**

- ◆ queue1 shares 50% of slots to be allocated =  $2 * 6 * 0.5 = 6$
- ◆ queue2 shares 30% of slots to be allocated =  $2 * 6 * 0.3 = 3.6 \rightarrow 4$
- ◆ queue3 shares 20% of slots to be allocated =  $2 * 6 * 0.2 = 2.4 \rightarrow 2$
- ◆ queue4 shares no slots with other queues

queue4 causes the total number of slots to be less than the total free and in use by the queue1, queue2, and queue3 that do belong to the pool. It is possible that the pool may get all its shares used up by queue4, and jobs from the pool will remain pending.

queue1, queue2, and queue3 belong to one pool, queue6, queue7, and queue8 belong to another pool, and queue4 and queue5 do not belong to any pool.

LSF orders the queues in the two pools from higher-priority queue to lower-priority queue (queue1 is highest and queue8 is lowest):

queue1 -> queue2 -> queue3 -> queue6 -> queue7 -> queue8

If the queue belongs to a pool, jobs are dispatched from the highest priority queue first. Queues that do not belong to any pool (queue4 and queue5) are merged into this ordered list according to their priority, but LSF dispatches as many jobs from the non-pool queues as it can:

queue1 -> queue2 -> queue3 -> queue4 -> queue5 -> queue6 -> queue7 -> queue8

## Configuring Slot Allocation per Queue

Configure as many pools as you need in `lsb.queues`.

### SLOT\_SHARE parameter

The `SLOT_SHARE` parameter represents the percentage of running jobs (job slots) in use from the queue. `SLOT_SHARE` must be greater than zero (0) and less than or equal to 100.

The sum of `SLOT_SHARE` for all queues in the pool does not need to be 100%. It can be more or less, depending on your needs.

### SLOT\_POOL parameter

The `SLOT_POOL` parameter is the name of the pool of job slots the queue belongs to. A queue can only belong to one pool. All queues in the pool must share the same set of hosts.

### Host job slot limit

The hosts used by the pool must have a maximum job slot limit, configured either in `lsb.hosts (MXJ)` or `lsb.resources (HOSTS and SLOTS)`.

## Configure slot allocation per queue

- 1 For each queue that uses queue-based fairshare, define the following in `lsb.queues`:
  - a `SLOT_SHARE`

## Configuring Slot Allocation per Queue

- b** `SLOT_POOL`
  - 2** Optional: Define the following in `lsb.queues` for each queue that uses queue-based fairshare:
    - a** `HOSTS` to list the hosts that can receive jobs from the queue  
If no hosts are defined for the queue, the default is all hosts.
- 
- TIP:** Hosts for queue-based fairshare cannot be in a host partition.
- 
- b** `PRIORITY` to indicate the priority of the queue.
  - 3** For each host used by the pool, define a maximum job slot limit, either in `lsb.hosts` (`MXJ`) or `lsb.resources` (`HOSTS` and `SLOTS`).

## Configure two pools

The following example configures pool A with three queues, with different shares, using the hosts in host group groupA:

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY = 50
SLOT_POOL = poolA
SLOT_SHARE = 50
HOSTS = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue2
PRIORITY = 48
SLOT_POOL = poolA
SLOT_SHARE = 30
HOSTS = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY = 46
SLOT_POOL = poolA
SLOT_SHARE = 20
HOSTS = groupA
...
End Queue
```

The following configures a pool named `poolB`, with three queues with equal shares, using the hosts in host group `groupB`:

```
Begin Queue
QUEUE_NAME = queue4
PRIORITY = 44
SLOT_POOL = poolB
SLOT_SHARE = 30
HOSTS = groupB
...
End Queue
```



```

Begin Queue
QUEUE_NAME = queue5
PRIORITY = 43
SLOT_POOL = poolB
SLOT_SHARE = 30
HOSTS = groupB
...
End Queue

Begin Queue
QUEUE_NAME = queue6
PRIORITY = 42
SLOT_POOL = poolB
SLOT_SHARE = 30
HOSTS = groupB
...
End Queue

```

## View Queue-based Fairshare Allocations

### View configured job slot share

- 1 Use `bqueues -l` to show the job slot share (SLOT\_SHARE) and the hosts participating in the share pool (SLOT\_POOL):

```
QUEUE: queue1
```

#### PARAMETERS/STATISTICS

```

PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
 50 20 Open:Active - - - - 0 0 0 0 0
Interval for a host to accept two jobs is 0 seconds

```

#### STACKLIMIT MEMLIMIT

```
2048 K 5000 K
```

#### SCHEDULING PARAMETERS

```

 r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

```

```

 cpuspeed bandwidth
loadSched - -
loadStop - -

```

```
USERS: all users
```

```
HOSTS: groupA/
```

```
SLOT_SHARE: 50%
```

```
SLOT_POOL: poolA
```

## View slot allocation of running jobs

- 1 Use `bhosts`, `bmgroup`, and `bqueues` to verify how LSF maintains the configured percentage of running jobs in each queue.

The queues configurations above use the following hosts groups:

```
bmgroup -r
GROUP_NAME HOSTS
groupA hosta hostb hostc
groupB hostd hoste hostf
```

Each host has a maximum job slot limit of 5, for a total of 15 slots available to be allocated in each group:

```
bhosts
HOST_NAME STATUS JL/U MAX NJOBS RUN SSUSP USUSP RSV
hosta ok - 5 5 5 0 0 0
hostb ok - 5 5 5 0 0 0
hostc ok - 5 5 5 0 0 0
hostd ok - 5 5 5 0 0 0
hoste ok - 5 5 5 0 0 0
hostf ok - 5 5 5 0 0 0
```

Pool named `poolA` contains `queue1`, `queue2`, and `queue3`. `poolB` contains `queue4`, `queue5`, and `queue6`. The `bqueues` command shows the number of running jobs in each queue:

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
queue1 50 Open:Active - - - - 492 484 8 0
queue2 48 Open:Active - - - - 500 495 5 0
queue3 46 Open:Active - - - - 498 496 2 0
queue4 44 Open:Active - - - - 985 980 5 0
queue5 43 Open:Active - - - - 985 980 5 0
queue6 42 Open:Active - - - - 985 980 5 0
```

As a result: `queue1` has a 50% share and can run 8 jobs; `queue2` has a 30% share and can run 5 jobs; `queue3` has a 20% share and is entitled 3 slots, but since the total number of slots available must be 15, it can run 2 jobs; `queue4`, `queue5`, and `queue6` all share 30%, so 5 jobs are running in each queue.

## Typical Slot Allocation Scenarios

### 3 queues with `SLOT_SHARE` 50%, 30%, 20%, with 15 job slots

This scenario has three phases:

- 1 All three queues have jobs running, and LSF assigns the number of slots to queues as expected: 8, 5, 2. Though queue `Genova` deserves 3 slots, the total slot assignment must be 15, so `Genova` is allocated only 2 slots:

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
Roma 50 Open:Active - - - - 1000 992 8 0
Verona 48 Open:Active - - - - 995 990 5 0
Genova 48 Open:Active - - - - 996 994 2 0
```

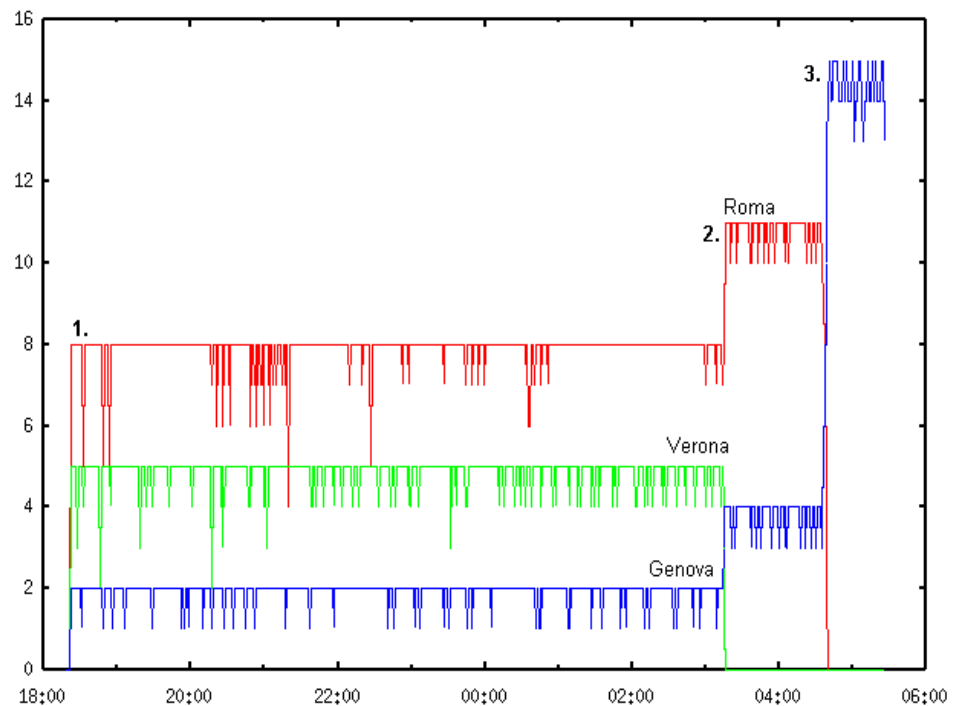
- 2 When queue *Verona* has done its work, queues *Roma* and *Genova* get their respective shares of 8 and 3. This leaves 4 slots to be redistributed to queues according to their shares: 50% (2 slots) to *Roma*, 20% (1 slot) to *Genova*. The one remaining slot is assigned to queue *Roma* again:

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
Roma 50 Open:Active - - - - 231 221 11 0
Verona 48 Open:Active - - - - 0 0 0 0
Genova 48 Open:Active - - - - 496 491 4 0
```

- 3 When queues *Roma* and *Verona* have no more work to do, *Genova* can use all the available slots in the cluster:

```
bqueues
QUEUE_NAME PRIO STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
Roma 50 Open:Active - - - - 0 0 0 0
Verona 48 Open:Active - - - - 0 0 0 0
Genova 48 Open:Active - - - - 475 460 15 0
```

The following figure illustrates phases 1, 2, and 3:



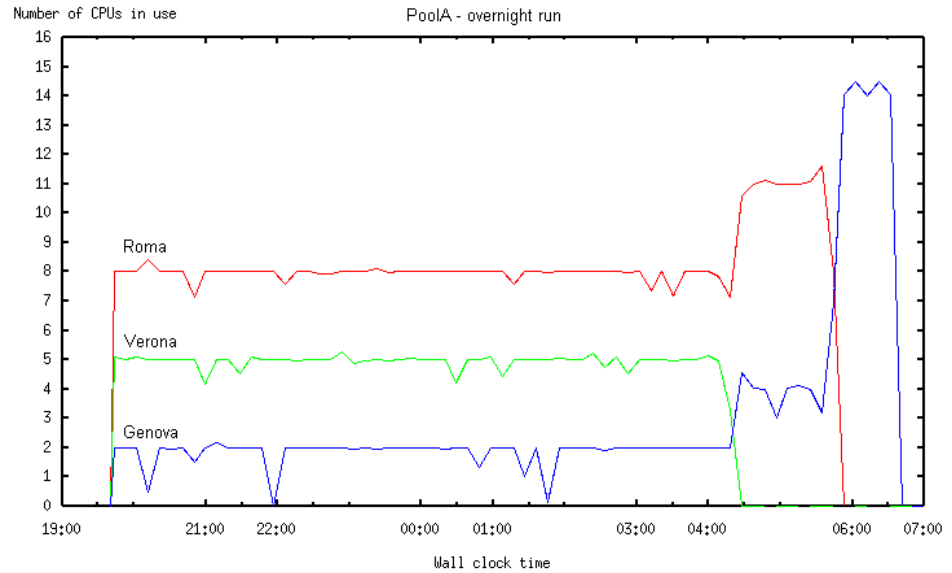
## 2 pools, 30 job slots, and 2 queues out of any pool

- ◆ poolA uses 15 slots and contains queues *Roma* (50% share, 8 slots), *Verona* (30% share, 5 slots), and *Genova* (20% share, 2 remaining slots to total 15).
- ◆ poolB with 15 slots containing queues *Pisa* (30% share, 5 slots), *Venezia* (30% share, 5 slots), and *Bologna* (30% share, 5 slots).
- ◆ Two other queues *Milano* and *Parma* do not belong to any pool, but they can use the hosts of poolB. The queues from *Milano* to *Bologna* all have the same priority.

## Typical Slot Allocation Scenarios

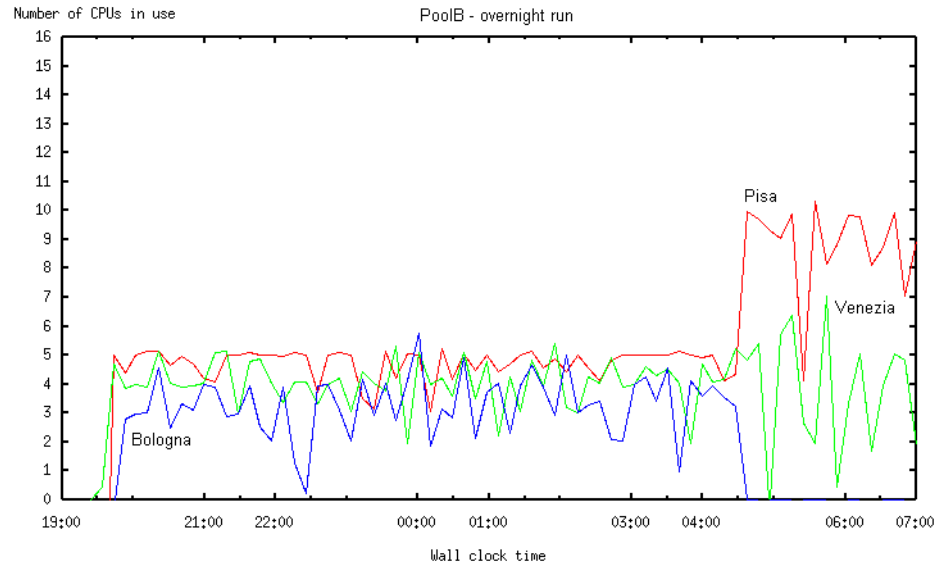
The queues Milano and Parma run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

| QUEUE_NAME | PRIO | STATUS      | MAX | JL/U | JL/P | JL/H | NJOBS | PEND | RUN | SUSP |
|------------|------|-------------|-----|------|------|------|-------|------|-----|------|
| Roma       | 50   | Open:Active | -   | -    | -    | -    | 1000  | 992  | 8   | 0    |
| Verona     | 48   | Open:Active | -   | -    | -    | -    | 1000  | 995  | 5   | 0    |
| Genova     | 48   | Open:Active | -   | -    | -    | -    | 1000  | 998  | 2   | 0    |
| Pisa       | 44   | Open:Active | -   | -    | -    | -    | 1000  | 995  | 5   | 0    |
| Milano     | 43   | Open:Active | -   | -    | -    | -    | 2     | 2    | 0   | 0    |
| Parma      | 43   | Open:Active | -   | -    | -    | -    | 2     | 2    | 0   | 0    |
| Venezia    | 43   | Open:Active | -   | -    | -    | -    | 1000  | 995  | 5   | 0    |
| Bologna    | 43   | Open:Active | -   | -    | -    | -    | 1000  | 995  | 5   | 0    |



When Milano and Parma have jobs, their higher priority reduces the share of slots free and in use by Venezia and Bologna:

| QUEUE_NAME | PRIO | STATUS      | MAX | JL/U | JL/P | JL/H | NJOBS | PEND | RUN | SUSP |
|------------|------|-------------|-----|------|------|------|-------|------|-----|------|
| Roma       | 50   | Open:Active | -   | -    | -    | -    | 992   | 984  | 8   | 0    |
| Verona     | 48   | Open:Active | -   | -    | -    | -    | 993   | 990  | 3   | 0    |
| Genova     | 48   | Open:Active | -   | -    | -    | -    | 996   | 994  | 2   | 0    |
| Pisa       | 44   | Open:Active | -   | -    | -    | -    | 995   | 990  | 5   | 0    |
| Milano     | 43   | Open:Active | -   | -    | -    | -    | 10    | 7    | 3   | 0    |
| Parma      | 43   | Open:Active | -   | -    | -    | -    | 11    | 8    | 3   | 0    |
| Venezia    | 43   | Open:Active | -   | -    | -    | -    | 995   | 995  | 2   | 0    |
| Bologna    | 43   | Open:Active | -   | -    | -    | -    | 995   | 995  | 2   | 0    |



## Round-robin slot distribution — 13 queues and 2 pools

- ◆ Pool `poolA` has 3 hosts each with 7 slots for a total of 21 slots to be shared. The first 3 queues are part of the pool `poolA` sharing the CPUs with proportions 50% (11 slots), 30% (7 slots) and 20% (3 remaining slots to total 21 slots).
- ◆ The other 10 queues belong to pool `poolB`, which has 3 hosts each with 7 slots for a total of 21 slots to be shared. Each queue has 10% of the pool (3 slots).

The initial slot distribution looks like this:

| bqueues    |      |             |     |      |      |      |       |      |     |      |  |
|------------|------|-------------|-----|------|------|------|-------|------|-----|------|--|
| QUEUE_NAME | PRIO | STATUS      | MAX | JL/U | JL/P | JL/H | NJOBS | PEND | RUN | SUSP |  |
| Roma       | 50   | Open:Active | -   | -    | -    | -    | 15    | 6    | 11  | 0    |  |
| Verona     | 48   | Open:Active | -   | -    | -    | -    | 25    | 18   | 7   | 0    |  |
| Genova     | 47   | Open:Active | -   | -    | -    | -    | 460   | 455  | 3   | 0    |  |
| Pisa       | 44   | Open:Active | -   | -    | -    | -    | 264   | 261  | 3   | 0    |  |
| Milano     | 43   | Open:Active | -   | -    | -    | -    | 262   | 259  | 3   | 0    |  |
| Parma      | 42   | Open:Active | -   | -    | -    | -    | 260   | 257  | 3   | 0    |  |
| Bologna    | 40   | Open:Active | -   | -    | -    | -    | 260   | 257  | 3   | 0    |  |
| Sora       | 40   | Open:Active | -   | -    | -    | -    | 261   | 258  | 3   | 0    |  |
| Ferrara    | 40   | Open:Active | -   | -    | -    | -    | 258   | 255  | 3   | 0    |  |
| Napoli     | 40   | Open:Active | -   | -    | -    | -    | 259   | 256  | 3   | 0    |  |
| Livorno    | 40   | Open:Active | -   | -    | -    | -    | 258   | 258  | 0   | 0    |  |
| Palermo    | 40   | Open:Active | -   | -    | -    | -    | 256   | 256  | 0   | 0    |  |
| Venezia    | 4    | Open:Active | -   | -    | -    | -    | 255   | 255  | 0   | 0    |  |

Initially, queues `Livorno`, `Palermo`, and `Venezia` in `poolB` are not assigned any slots because the first 7 higher priority queues have used all 21 slots available for allocation.

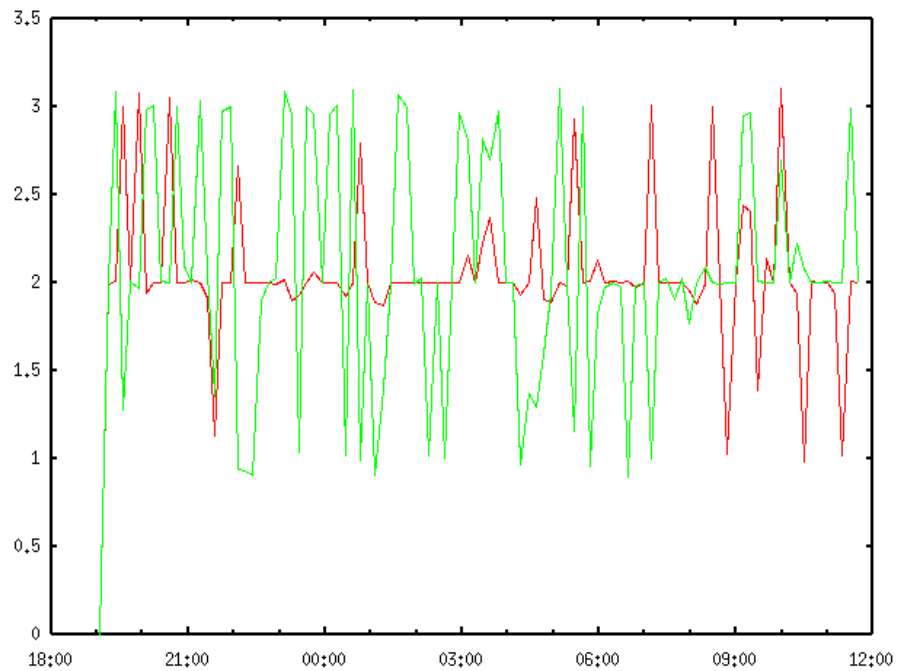
As jobs run and each queue accumulates used slots, LSF favors queues that have not run jobs yet. As jobs finish in the first 7 queues of `poolB`, slots are redistributed to the other queues that originally had no jobs (queues `Livorno`, `Palermo`, and `Venezia`). The total slot count remains 21 in all queues in `poolB`.

| bqueues    |      |             |     |      |      |      |       |      |     |      |  |
|------------|------|-------------|-----|------|------|------|-------|------|-----|------|--|
| QUEUE_NAME | PRIO | STATUS      | MAX | JL/U | JL/P | JL/H | NJOBS | PEND | RUN | SUSP |  |
| Roma       | 50   | Open:Active | -   | -    | -    | -    | 15    | 6    | 9   | 0    |  |

## Typical Slot Allocation Scenarios

|         |    |             |   |   |   |   |     |     |   |   |
|---------|----|-------------|---|---|---|---|-----|-----|---|---|
| V       | 48 | Open:Active | - | - | - | - | 25  | 18  | 7 | 0 |
| Genova  | 47 | Open:Active | - | - | - | - | 460 | 455 | 5 | 0 |
| Pisa    | 44 | Open:Active | - | - | - | - | 263 | 261 | 2 | 0 |
| Milano  | 43 | Open:Active | - | - | - | - | 261 | 259 | 2 | 0 |
| Parma   | 42 | Open:Active | - | - | - | - | 259 | 257 | 2 | 0 |
| Bologna | 40 | Open:Active | - | - | - | - | 259 | 257 | 2 | 0 |
| Sora    | 40 | Open:Active | - | - | - | - | 260 | 258 | 2 | 0 |
| Ferrara | 40 | Open:Active | - | - | - | - | 257 | 255 | 2 | 0 |
| Napoli  | 40 | Open:Active | - | - | - | - | 258 | 256 | 2 | 0 |
| Livorno | 40 | Open:Active | - | - | - | - | 258 | 256 | 2 | 0 |
| Palermo | 40 | Open:Active | - | - | - | - | 256 | 253 | 3 | 0 |
| Venezia | 4  | Open:Active | - | - | - | - | 255 | 253 | 2 | 0 |

The following figure illustrates the round-robin distribution of slot allocations between queues Livorno and Palermo:



## How LSF rebalances slot usage

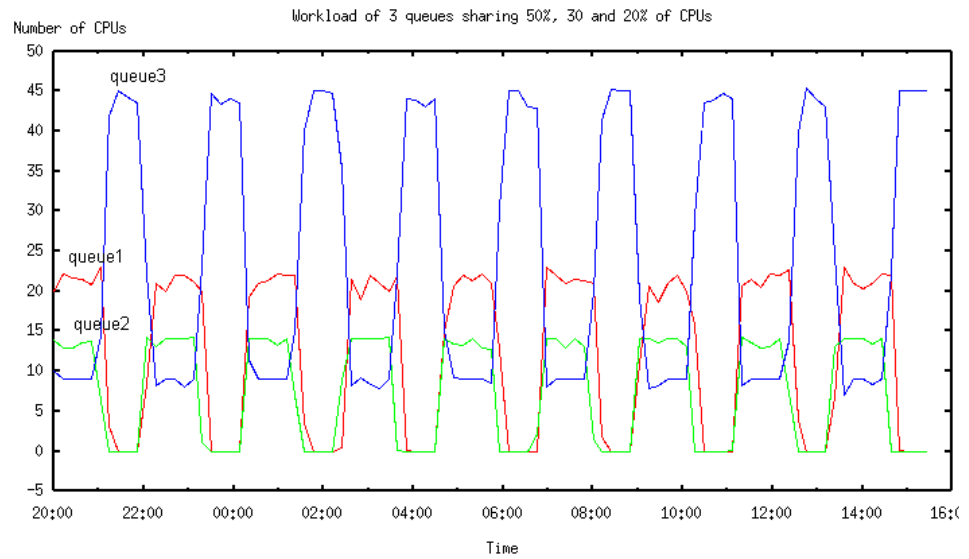
In the following examples, job runtime is not equal, but varies randomly over time.

### 3 queues in one pool with 50%, 30%, 20% shares

A pool configures 3 queues:

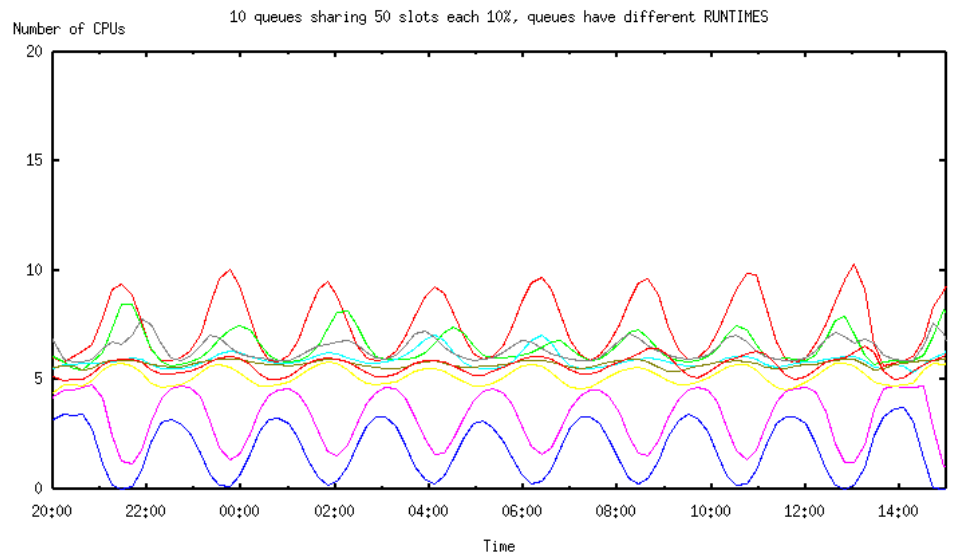
- ◆ queue1 50% with short-running jobs
- ◆ queue2 20% with short-running jobs
- ◆ queue3 30% with longer running jobs

As queue1 and queue2 finish their jobs, the number of jobs in queue3 expands, and as queue1 and queue2 get more work, LSF rebalances the usage:



10 queues sharing  
10% each of 50  
slots

In this example, `queue1` (the curve with the highest peaks) has the longer running jobs and so has less accumulated slots in use over time. LSF accordingly rebalances the load when all queues compete for jobs to maintain a configured 10% usage share.



## Using Historical and Committed Run Time

By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.

In some cases this can interfere with fairshare scheduling if two users who have the same priority and the same number of shares submit jobs at the same time.

To avoid these problems, you can modify the dynamic priority calculation by using either or both of the following weighting factors:

- ◆ Historical run time decay
- ◆ Committed run time

## Historical run time decay

By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority. LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.

### Configure historical run time

- 1 Specify `ENABLE_HIST_RUN_TIME=Y` in `lsb.params`.

Historical run time is added to the calculation of the dynamic priority so that the formula becomes the following:

$$\text{dynamic priority} = \text{number\_shares} / (\text{cpu\_time} * \text{CPU\_TIME\_FACTOR} + (\text{historical\_run\_time} + \text{run\_time}) * \text{RUN\_TIME\_FACTOR} + (1 + \text{job\_slots}) * \text{RUN\_JOB\_FACTOR})$$

*historical\_run\_time*—(measured in hours) of finished jobs accumulated in the user's share account file. LSF calculates the historical run time using the actual run time of finished jobs and a decay factor such that 1 hour of recently-used run time decays to 0.1 hours after an interval of time specified by `HIST_HOURS` in `lsb.params` (5 hours by default).

### How mbatchd reconfiguration and restart affects historical run time

After restarting or reconfiguring `mbatchd`, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from `mbatchd` before the restart. `mbatchd` restart only reads recently finished jobs from `lsb.events`, according to the value of `CLEAN_PERIOD` in `lsb.params`. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

### Example

The following fairshare parameters are configured in `lsb.params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR = 0
RUN_TIME_FACTOR = 1
```

Note that in this configuration, only run time is considered in the calculation of dynamic priority. This simplifies the formula to the following:

$$\text{dynamic priority} = \text{number\_shares} / (\text{run\_time} * \text{RUN\_TIME\_FACTOR})$$

Without the historical run time, the dynamic priority increases suddenly as soon as the job finishes running because the run time becomes zero, which gives no chance for jobs pending for other users to start.

When historical run time is included in the priority calculation, the formula becomes:

$$\text{dynamic priority} = \text{number\_shares} / (\text{historical\_run\_time} + \text{run\_time}) * \text{RUN\_TIME\_FACTOR}$$

Now the dynamic priority increases gradually as the historical run time decays over time.



## Committed run time weighting factor

*Committed run time* is the run time requested at job submission with the `-W` option of `bsub`, or in the queue configuration with the `RUNLIMIT` parameter. By default, committed run time does not affect the dynamic priority.

While the job is running, the actual run time is subtracted from the committed run time. The user's dynamic priority decreases *immediately* to its lowest expected value, and is maintained at that value until the job finishes. Job run time is accumulated as usual, and historical run time, if any, is decayed.

When the job finishes, the committed run time is set to zero and the actual run time is added to the historical run time for future use. The dynamic priority increases gradually until it reaches its maximum value.

Providing a weighting factor in the run time portion of the dynamic priority calculation prevents a "job dispatching burst" where one user monopolizes job slots because of the latency in computing run time.

### Configure committed run time

- 1 Set a value for the `COMMITTED_RUN_TIME_FACTOR` parameter in `lsb.params`. You should also specify a `RUN_TIME_FACTOR`, to prevent the user's dynamic priority from increasing as the run time increases.

If you have also enabled the use of historical run time, the dynamic priority is calculated according to the following formula:

$$\text{dynamic priority} = \text{number\_shares} / (\text{cpu\_time} * \text{CPU\_TIME\_FACTOR} + (\text{historical\_run\_time} + \text{run\_time}) * \text{RUN\_TIME\_FACTOR} + (\text{committed\_run\_time} - \text{run\_time}) * \text{COMMITTED\_RUN\_TIME\_FACTOR} + (1 + \text{job\_slots}) * \text{RUN\_JOB\_FACTOR})$$

*committed\_run\_time*—The run time requested at job submission with the `-W` option of `bsub`, or in the queue configuration with the `RUNLIMIT` parameter. This calculation measures the committed run time in hours.

In the calculation of a user's dynamic priority, `COMMITTED_RUN_TIME_FACTOR` determines the relative importance of the committed run time in the calculation. If the `-W` option of `bsub` is not specified at job submission and a `RUNLIMIT` has not been set for the queue, the committed run time is not considered.

`COMMITTED_RUN_TIME_FACTOR` can be any positive value between 0.0 and 1.0. The default value is 0.0. As the value of `COMMITTED_RUN_TIME_FACTOR` approaches 1.0, more weight is given to the committed run time in the calculation of the dynamic priority.

### Limitation

If you use queue-level fairshare, and a running job has a committed run time, you should not switch that job to or from a fairshare queue (using `bswitch`). The fairshare calculations will not be correct.

### Run time displayed by `bqueues` and `bhpart`

The run time displayed by `bqueues` and `bhpart` is the sum of the actual, accumulated run time and the historical run time, but does not include the committed run time.

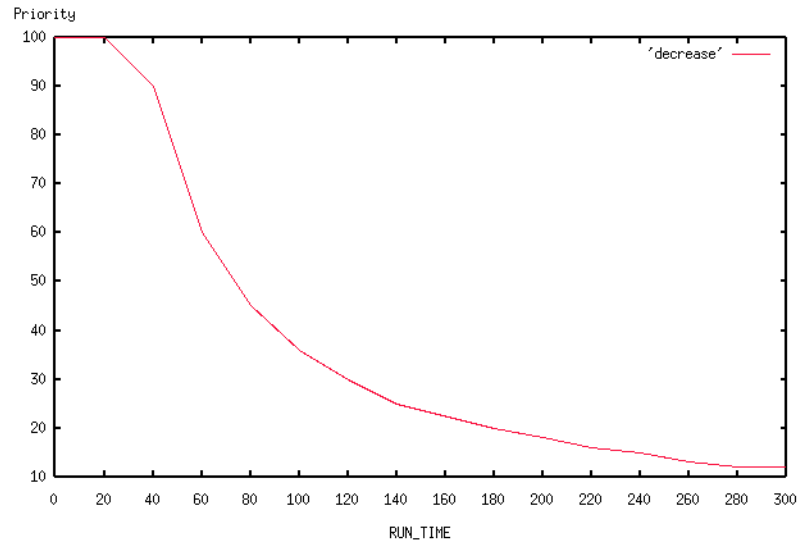
## Using Historical and Committed Run Time

### Example

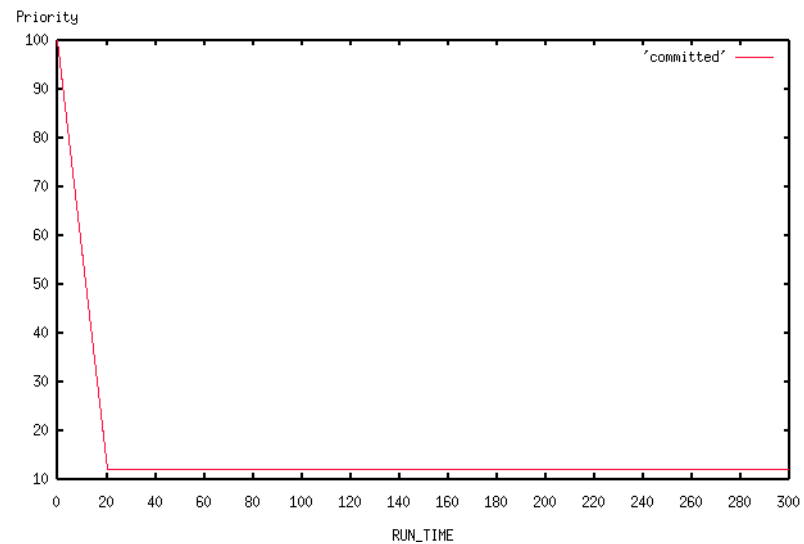
The following fairshare parameters are configured in `lsb.params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR = 0
RUN_TIME_FACTOR = 1
COMMITTED_RUN_TIME_FACTOR = 1
```

Without a committed run time factor, dynamic priority for the job owner drops gradually while a job is running:



When a committed run time factor is included in the priority calculation, the dynamic priority drops as soon as the job is dispatched, rather than gradually dropping as the job runs:



## Users Affected by Multiple Fairshare Policies

If you belong to multiple user groups, which are controlled by different fairshare policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.

To restrict the number of fairshare policies that will affect your job, submit your job and specify a single user group that your job will belong to, for the purposes of fairshare scheduling. LSF will not attempt to dispatch this job unless the group you specified is the highest priority user. If you become the highest priority user because of some other share assignment, another one of your jobs might be dispatched, but not this one.

### Submit a job and specify a user group

- 1 To associate a job with a user group for the purposes of fairshare scheduling, use `bsub -G` and specify a group that you belong to. If you use hierarchical fairshare, you must specify a group that does not contain any subgroups.

#### Example

User1 shares resources with groupA and groupB. User1 is also a member of groupA, but not any other groups.

User1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User1 or GroupA has highest dynamic share priority.

User1 submits a job and associates the job with GroupA:

```
bsub -G groupA sleep 100
```

If User1 is the highest priority user, this job will not be considered.

- ◆ User1 can only associate the job with a group that he is a member of.
- ◆ User1 cannot associate the job with his individual user account, because `bsub -G` only accepts group names.

#### Example with hierarchical fairshare

In the share tree, User1 shares resources with GroupA at the top level. GroupA has 2 subgroups, B and C. GroupC has 1 subgroup, GroupD. User1 also belongs to GroupB and GroupC.

User1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User1, GroupB, or GroupC has highest dynamic share priority.

User1 submits a job and associates the job with GroupB:

```
bsub -G groupB sleep 100
```

If User1 or GroupC is the highest priority user, this job will not be considered.

## Ways to Configure Fairshare

- ◆ User1 cannot associate the job with GroupC, because GroupC includes a subgroup.
- ◆ User1 cannot associate the job with his individual user account, because `bsub -G` only accepts group names.

## Ways to Configure Fairshare

### Global fairshare

Global fairshare balances resource usage across the entire cluster according to one single fairshare policy. Resources used in one queue affect job dispatch order in another queue.

If two users compete for resources, their dynamic share priority is the same in every queue.

### Configure global fairshare

- 1 To configure global fairshare, you must use host partition fairshare. Use the keyword `all` to configure a single partition that includes all the hosts in the cluster.

```
Begin HostPartition
HPART_NAME =GlobalPartition
HOSTS = all
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

### Chargeback fairshare

Chargeback fairshare lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.

If two users compete for resources, the most important user is entitled to more resources.

### Configure chargeback fairshare

- 1 To configure chargeback fairshare, put competing users in separate user groups and assign a fair number of shares to each group.

### Example

Suppose two departments contributed to the purchase of a large system. The engineering department contributed 70 percent of the cost, and the accounting department 30 percent. Each department wants to get their money's worth from the system.

- 1 Define 2 user groups in `lsb.users`, one listing all the engineers, and one listing all the accountants.

```
Begin UserGroup
Group_Name Group_Member
eng_users (user6 user4)
acct_users (user2 user5)
End UserGroup
```

- 2 Configure a host partition for the host, and assign the shares appropriately.

```
Begin HostPartition
HPART_NAME = big_servers
HOSTS = hostH
USER_SHARES = [eng_users, 7] [acct_users, 3]
End HostPartition
```

---

## Equal Share

Equal share balances resource usage equally between users.

---

### Configure equal share

- 1 To configure equal share, use the keyword `default` to define an equal share for every user.

```
Begin HostPartition
HPART_NAME = equal_share_partition
HOSTS = all
USER_SHARES = [default, 1]
End HostPartition
```

---

## Priority user and static priority fairshare

There are two ways to configure fairshare so that a more important user's job always overrides the job of a less important user, regardless of resource use.

- ◆ **Priority User Fairshare:** Dynamic priority is calculated as usual, but more important and less important users are assigned a drastically different number of shares, so that resource use has virtually no effect on the dynamic priority: the user with the overwhelming majority of shares always goes first. However, if two users have a similar or equal number of shares, their resource use still determines which of them goes first. This is useful for isolating a group of high-priority or low-priority users, while allowing other fairshare policies to operate as usual most of the time.
- ◆ **Static Priority Fairshare:** Dynamic priority is no longer dynamic, because resource use is ignored. The user with the most shares always goes first. This is useful to configure multiple users in a descending order of priority.

## Configure priority user fairshare

A queue is shared by key users and other users.

Priority user fairshare gives priority to important users, so their jobs override the jobs of other users. You can still use fairshare policies to balance resources among each group of users.

If two users compete for resources, and one of them is a priority user, the priority user's job always runs first.

---

- 1 Define a user group for priority users in `lsb.users`, naming it accordingly. For example, `key_users`.

- 2 Configure fairshare and assign the overwhelming majority of shares to the key users:

```
Begin Queue
QUEUE_NAME = production
FAIRSHARE = USER_SHARES[[key_users@, 2000] [others, 1]]
...
End Queue
```

In the above example, key users have 2000 shares each, while other users together have only 1 share. This makes it virtually impossible for other users' jobs to get dispatched unless none of the users in the `key_users` group has jobs waiting to run.

If you want the same fairshare policy to apply to jobs from all queues, configure host partition fairshare in a similar way.

---

## Configure static priority fairshare

Static priority fairshare assigns resources to the user with the most shares. Resource usage is ignored.

---

- 1 To implement static priority fairshare, edit `lsb.params` and set all the weighting factors used in the dynamic priority formula to 0 (zero).
    - ◆ Set `CPU_TIME_FACTOR` to 0
    - ◆ Set `RUN_TIME_FACTOR` to 0
    - ◆ Set `RUN_JOB_FACTOR` to 0
    - ◆ Set `COMMITTED_RUN_TIME_FACTOR` to 0
- 

If two users compete for resources, the most important user's job always runs first.

## Goal-Oriented SLA-Driven Scheduling

### Contents

- ◆ [Using Goal-Oriented SLA Scheduling](#) on page 327
- ◆ [Configuring Service Classes for SLA Scheduling](#) on page 330
- ◆ [View Information about SLAs and Service Classes](#) on page 332
- ◆ [Understanding Service Class Behavior](#) on page 336
- ◆ [EGO-enabled SLA scheduling](#) on page 341

### Using Goal-Oriented SLA Scheduling

Goal-oriented SLA scheduling policies help you configure your workload so that your jobs are completed on time and reduce the risk of missed deadlines. They enable you to focus on the “what and when” of your projects, not the low-level details of “how” resources need to be allocated to satisfy various workloads.

### Service-level agreements in LSF

A *service-level agreement* (SLA) defines how a service is delivered and the parameters for the delivery of a service. It specifies what a service provider and a service recipient agree to, defining the relationship between the provider and recipient with respect to a number of issues, among them:

- ◆ Services to be delivered
- ◆ Performance
- ◆ Tracking and reporting
- ◆ Problem management

An SLA in LSF is a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users. The SLA scheduling policy defines how many jobs should be run from each SLA to meet the configured goals.

## Service classes

SLA definitions consist of service-level goals that are expressed in individual *service classes*. A service class is the actual configured policy that sets the service-level goals for the LSF system. The SLA defines the workload (jobs or other services) and users that need the work done, while the service class that addresses the SLA defines individual goals, and a time window when the service class is active.

## Service-level goals

You configure the following kinds of goals:

### Deadline goals

A specified number of jobs should be completed within a specified time window. For example, run all jobs submitted over a weekend.

### Velocity goals

Expressed as concurrently running jobs. For example: maintain 10 running jobs between 9:00 a.m. and 5:00 p.m. Velocity goals are well suited for short jobs (run time less than one hour). Such jobs leave the system quickly, and configuring a velocity goal ensures a steady flow of jobs through the system.

### Throughput goals

Expressed as number of finished jobs per hour. For example: finish 15 jobs per hour between the hours of 6:00 p.m. and 7:00 a.m. Throughput goals are suitable for medium to long running jobs. These jobs stay longer in the system, so you typically want to control their rate of completion rather than their flow.

### Combining different types of goals

You might want to set velocity goals to maximize quick work during the day, and set deadline and throughput goals to manage longer running work on nights and over weekends.

## How service classes perform goal-oriented scheduling

Goal-oriented scheduling makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses. The decisions of a service class are considered first before any queue or host partition decisions. Limits are still enforced with respect to lower level scheduling objects like queues, hosts, and users.

### Optimum number of running jobs

As jobs are submitted, LSF determines the optimum number of job slots (or concurrently running jobs) needed for the service class to meet its service-level goals. LSF schedules a number of jobs at least equal to the optimum number of slots calculated for the service class.

LSF attempts to meet SLA goals in the most efficient way, using the optimum number of job slots so that other service classes or other types of work in the cluster can still progress. For example, in a service class that defines a deadline goal, LSF spreads out the work over the entire time window for the goal, which avoids blocking other work by not allocating as many slots as possible at the beginning to finish earlier than the deadline.



## Submit jobs to a service class

You submit jobs to a service class as you would to a queue, except that a service class is a higher level scheduling policy that makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses.

The service class name where the job is to run is configured in `lsb.serviceclasses`. If the SLA does not exist or the user is not a member of the service class, the job is rejected.

Outside of the configured time windows, the SLA is not active, and LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with `-sla`.

- 
- 1 Run `bsub -sla service_class_name` to submit a job to a service class for SLA-driven scheduling.  
`bsub -W 15 -sla Kyuquot sleep 100`  
 submits the UNIX command `sleep` together with its argument 100 as a job to the service class named `Kyuquot`.
- 

## Submitting with a run limit

You should submit your jobs with a run time limit at the job level (`-W` option), the application level (RUNLIMIT parameter in the application definition in `lsb.applications`), or the queue level (RUNLIMIT parameter in the queue definition in `lsb.queues`). You can also submit the job with a run time estimate defined at the application level (RUNTIME parameter in `lsb.applications`) instead of or in conjunction with the run time limit.

The following table describes how LSF uses the values that you provide for SLA-driven scheduling.

| If you specify...                                | And...                                                                            | Then...                                                                                                           |
|--------------------------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| A run time limit and a run time estimate         | The run time estimate is less than or equal to the run time limit                 | LSF uses the run time estimate to compute the optimum number of running jobs.                                     |
| A run time limit                                 | You do not specify a run time estimate, or the estimate is greater than the limit | LSF uses the run time limit to compute the optimum number of running jobs.                                        |
| A run time estimate                              | You do not specify a run time limit                                               | LSF uses the run time estimate to compute the optimum number of running jobs.                                     |
| Neither a run time limit nor a run time estimate |                                                                                   | LSF automatically adjusts the optimum number of running jobs according to the observed run time of finished jobs. |

## Modify SLA jobs (bmod)

- 1 Run `bmod -sla` to modify the service class a job is attached to, or to attach a submitted job to a service class. Run `bmod -slan` to detach a job from a service class:

```
bmod -sla Kyuquot 2307
```

Attaches job 2307 to the service class `Kyuquot`.

```
bmod -slan 2307
```

Detaches job 2307 from the service class `Kyuquot`.

You cannot:

- ◆ Use `-sla` with other `bmod` options
- ◆ Move job array elements from one service class to another, only entire job arrays
- ◆ Modify the service class of jobs already attached to a job group

If a default SLA is configured in `lsb.params`, `bmod -slan` moves the job to the default SLA. If the job is already attached to the default SLA, `bmod -slan` has no effect on that job.

## Configuring Service Classes for SLA Scheduling

Configure service classes in

`LSB_CONFDIR/cluster_name/configdir/lsb.serviceclasses`. Each service class is defined in a `ServiceClass` section.

Each service class section begins with the line `Begin ServiceClass` and ends with the line `End ServiceClass`. You must specify:

- ◆ A service class name
- ◆ At least one goal (deadline, throughput, or velocity) and a time window when the goal is active
- ◆ A service class priority

All other parameters are optional. You can configure as many service class sections as you need.

---

**IMPORTANT:** The name you use for your service classes cannot be the same as an existing host partition or user group name.

---

### User groups for service classes

You can control access to the SLA by configuring a user group for the service class. If LSF user groups are specified in `lsb.users`, each user in the group can submit jobs to this service class. If a group contains a subgroup, the service class policy applies to each member in the subgroup recursively. The group can define fairshare among its members, and the SLA defined by the service class enforces the fairshare policy among the users in the user group configured for the SLA.

By default, all users in the cluster can submit jobs to the service class.

## Service class priority

A higher value indicates a higher priority, relative to other service classes. Similar to queue priority, service classes access the cluster resources in priority order.

LSF schedules jobs from one service class at a time, starting with the highest-priority service class. If multiple service classes have the same priority, LSF run all the jobs from these service classes in first-come, first-served order.

Service class priority in LSF is completely independent of the UNIX scheduler's priority system for time-sharing processes. In LSF, the NICE parameter is used to set the UNIX time-sharing priority for batch jobs.

## Service class configuration examples

- ◆ The service class `Uclulet` defines one deadline goal that is active during working hours between 8:30 AM and 4:00 PM. All jobs in the service class should complete by the end of the specified time window. Outside of this time window, the SLA is inactive and jobs are scheduled without any goal being enforced:

```
Begin ServiceClass
NAME = Uclulet
PRIORITY = 20
GOALS = [DEADLINE timeWindow (8:30-16:00)]
DESCRIPTION = "working hours"
End ServiceClass
```

- ◆ The service class `Nanaimo` defines a deadline goal that is active during the weekends and at nights.

```
Begin ServiceClass
NAME = Nanaimo
PRIORITY = 20
GOALS = [DEADLINE timeWindow (5:18:00-1:8:30 20:00-8:30)]
DESCRIPTION = "weekend nighttime regression tests"
End ServiceClass
```

- ◆ The service class `Inuvik` defines a throughput goal of 6 jobs per hour that is always active:

```
Begin ServiceClass
NAME = Inuvik
PRIORITY = 20
GOALS = [THROUGHPUT 6 timeWindow ()]
DESCRIPTION = "constant throughput"
End ServiceClass
```

---

**TIP:** To configure a time window that is always open, use the `timeWindow` keyword with empty parentheses.

---

- ◆ The service class `Tofino` defines two velocity goals in a 24 hour period. The first goal is to have a maximum of 10 concurrently running jobs during business hours (9:00 a.m. to 5:00 p.m.). The second goal is a maximum of 30 concurrently running jobs during off-hours (5:30 p.m. to 8:30 a.m.)

```
Begin ServiceClass
NAME = Tofino
PRIORITY = 20
GOALS = [VELOCITY 10 timeWindow (9:00-17:00)] \
```

## View Information about SLAs and Service Classes

```
[VELOCITY 30 timeWindow (17:30-8:30)]
DESCRIPTION = "day and night velocity"
End ServiceClass
```

- ◆ The service class `Kyuquot` defines a velocity goal that is active during working hours (9:00 a.m. to 5:30 p.m.) and a deadline goal that is active during off-hours (5:30 p.m. to 9:00 a.m.) Only users `user1` and `user2` can submit jobs to this service class.

```
Begin ServiceClass
NAME = Kyuquot
PRIORITY = 23
USER_GROUP = user1 user2
GOALS = [VELOCITY 8 timeWindow (9:00-17:30)] \
 [DEADLINE timeWindow (17:30-9:00)]
DESCRIPTION = "Daytime/Nighttime SLA"
End ServiceClass
```

- ◆ The service class `Tevere` defines a combination similar to `Kyuquot`, but with a deadline goal that takes effect overnight and on weekends. During the working hours in weekdays the velocity goal favors a mix of short and medium jobs.

```
Begin ServiceClass
NAME = Tevere
PRIORITY = 20
GOALS = [VELOCITY 100 timeWindow (9:00-17:00)] \
 [DEADLINE timeWindow (17:30-8:30 5:17:30-1:8:30)]
DESCRIPTION = "nine to five"
End ServiceClass
```

## View Information about SLAs and Service Classes

### Monitor the progress of an SLA (bsla)

- 1 Run `bsla` to display the properties of service classes configured in `lsb.serviceclasses` and dynamic information about the state of each configured service class.

### Examples

- ◆ One velocity goal of service class `Tofino` is active and on time. The other configured velocity goal is inactive.

```
bsla
SERVICE CLASS NAME: Tofino
-- day and night velocity
PRIORITY: 20

GOAL: VELOCITY 30
ACTIVE WINDOW: (17:30-8:30)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: VELOCITY 10
ACTIVE WINDOW: (9:00-17:00)
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 300   | 280  | 10  | 0     | 0     | 10     |

- ◆ The deadline goal of service class `Uclulet` is not being met, and `bsla` displays status `Active:Delayed`:

```
bsla
SERVICE CLASS NAME: Uclulet
-- working hours
PRIORITY: 20

GOAL: DEADLINE
ACTIVE WINDOW: (8:30-19:00)
STATUS: Active:Delayed
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
ESTIMATED FINISH TIME: (Tue Oct 28 06:17)
OPTIMUM NUMBER OF RUNNING JOBS: 6
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 40    | 39   | 1   | 0     | 0     | 0      |

- ◆ The configured velocity goal of the service class `Kyuquot` is active and on time. The configured deadline goal of the service class is inactive.

```
bsla Kyuquot
SERVICE CLASS NAME: Kyuquot
-- Daytime/Nighttime SLA
PRIORITY: 23
USER_GROUP: user1 user2

GOAL: VELOCITY 8
ACTIVE WINDOW: (9:00-17:30)
STATUS: Active:On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: DEADLINE
ACTIVE WINDOW: (17:30-9:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 0     | 0    | 0   | 0     | 0     | 0      |

- ◆ The throughput goal of service class `Inuvik` is always active. `bsla` displays:
  - ◆ Status as active and on time
  - ◆ An optimum number of 5 running jobs to meet the goal
  - ◆ Actual throughput of 10 jobs per hour based on the last `CLEAN_PERIOD`

```
bsla Inuvik
SERVICE CLASS NAME: Inuvik
-- constant throughput
PRIORITY: 20

GOAL: THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 5
```

## View Information about SLAs and Service Classes

```
 NJOBS PEND RUN SSUSP USUSP FINISH
 110 95 5 0 0 10
```

## View jobs running in an SLA (bjobs)

- 1 Run `bjobs -sla` to display jobs running in a service class:

```
bjobs -sla Inuvik
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
136 user1 RUN normal hostA hostA sleep 100 Sep 28 13:24
137 user1 RUN normal hostA hostB sleep 100 Sep 28 13:25
```

Use `-sla` with `-g` to display job groups attached to a service class. Once a job group is attached to a service class, all jobs submitted to that group are subject to the SLA.

## Track historical behavior of an SLA (bacct)

- 1 Run `bacct` to display historical performance of a service class. For example, service classes `Inuvik` and `Tuktoyaktuk` configure throughput goals.

```
bsla
SERVICE CLASS NAME: Inuvik
-- throughput 6
PRIORITY: 20

GOAL: THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 5
```

```
 NJOBS PEND RUN SSUSP USUSP FINISH
 111 94 5 0 0 12
```

```

SERVICE CLASS NAME: Tuktoyaktuk
-- throughput 3
PRIORITY: 15
```

```
GOAL: THROUGHPUT 3
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 4.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 4
```

```
 NJOBS PEND RUN SSUSP USUSP FINISH
 104 96 4 0 0 4
```

These two service classes have the following historical performance. For SLA `Inuvik`, `bacct` shows a total throughput of 8.94 jobs per hour over a period of 20.58 hours:

```
bacct -sla Inuvik
```

Accounting information about jobs that are:

- submitted by users user1,
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on service classes Inuvik,

```

SUMMARY: (time unit: second)
Total number of done jobs: 183 Total number of exited jobs: 1
Total CPU time consumed: 40.0 Average CPU time consumed: 0.2
Maximum CPU time of a job: 0.3 Minimum CPU time of a job: 0.1
Total wait time in queues: 1947454.0
Average wait time in queue:10584.0
Maximum wait time in queue:18912.0 Minimum wait time in queue: 7.0
Average turnaround time: 12268 (seconds/job)
Maximum turnaround time: 22079 Minimum turnaround time: 1713
Average hog factor of a job: 0.00 (cpu time / turnaround time)
Maximum hog factor of a job: 0.00 Minimum hog factor of a job: 0.00
Total throughput: 8.94 (jobs/hour) during 20.58 hours
Beginning time: Oct 11 20:23 Ending time: Oct 12 16:58

```

**For SLA Tuktoyaktuk, bacct shows a total throughput of 4.36 jobs per hour over a period of 19.95 hours:**

**bacct -sla Tuktoyaktuk**

Accounting information about jobs that are:

- submitted by users user1,
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on service classes Tuktoyaktuk,

```

SUMMARY: (time unit: second)
Total number of done jobs: 87 Total number of exited jobs: 0
Total CPU time consumed: 18.0 Average CPU time consumed: 0.2
Maximum CPU time of a job: 0.3 Minimum CPU time of a job: 0.1
Total wait time in queues: 2371955.0
Average wait time in queue:27263.8
Maximum wait time in queue:39125.0 Minimum wait time in queue: 7.0
Average turnaround time: 30596 (seconds/job)
Maximum turnaround time: 44778 Minimum turnaround time: 3355
Average hog factor of a job: 0.00 (cpu time / turnaround time)
Maximum hog factor of a job: 0.00 Minimum hog factor of a job: 0.00
Total throughput: 4.36 (jobs/hour) during 19.95 hours
Beginning time: Oct 11 20:50 Ending time: Oct 12 16:47

```

**Because the run times are not uniform, both service classes actually achieve higher throughput than configured.**

## Understanding Service Class Behavior

### A simple deadline goal

The following service class configures an SLA with a simple deadline goal with a half hour time window.

```
Begin ServiceClass
NAME = Quadra
PRIORITY = 20
GOALS = [DEADLINE timeWindow (16:15-16:45)]
DESCRIPTION = short window
End ServiceClass
```

Six jobs submitted with a run time of 5 minutes each will use 1 slot for the half hour time window. `bsla` shows that the deadline can be met:

```
bsla Quadra
SERVICE CLASS NAME: Quadra
-- short window
PRIORITY: 20

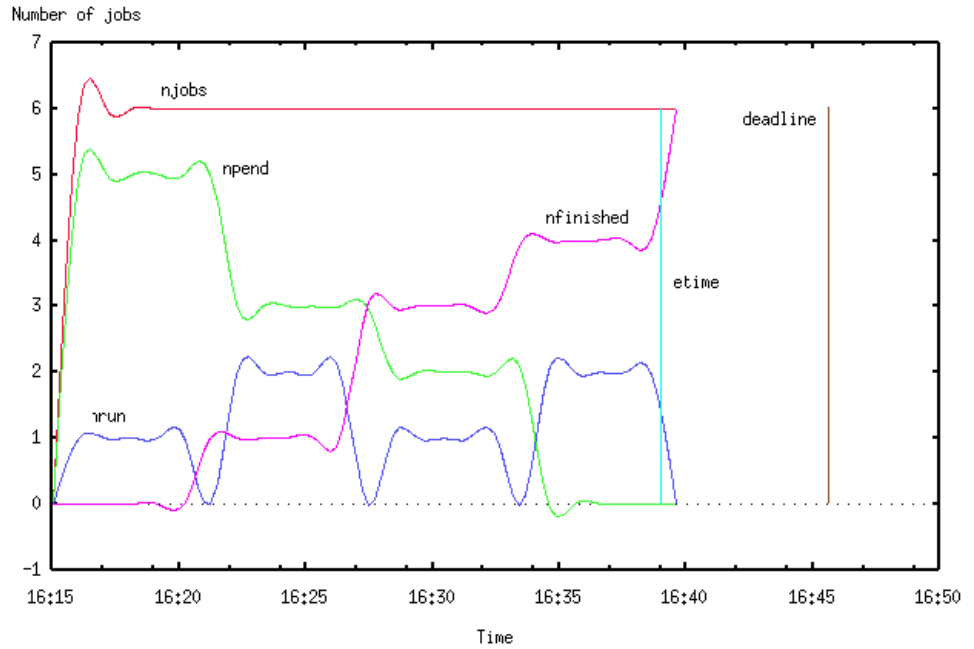
GOAL: DEADLINE
ACTIVE WINDOW: (16:15-16:45)
STATUS: Active:On time
ESTIMATED FINISH TIME: (Wed Jul 2 16:38)
OPTIMUM NUMBER OF RUNNING JOBS: 1
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 6     | 5    | 1   | 0     | 0     | 0      |

The following illustrates the progress of the SLA to the deadline. The optimum number of running jobs in the service class (`nrun`) is maintained at a steady rate of 1 job at a time until near the completion of the SLA.

When the finished job curve (`nfinished`) meets the total number of jobs curve (`njobs`) the deadline is met. All jobs are finished well ahead of the actual configured deadline, and the goal of the SLA was met.





### An overnight run with two service classes

bsla shows the configuration and status of two service classes Qualicum and Comox:

- ◆ Qualicum has a deadline goal with a time window which is active overnight:

```
bsla Qualicum
SERVICE CLASS NAME: Qualicum
PRIORITY: 23

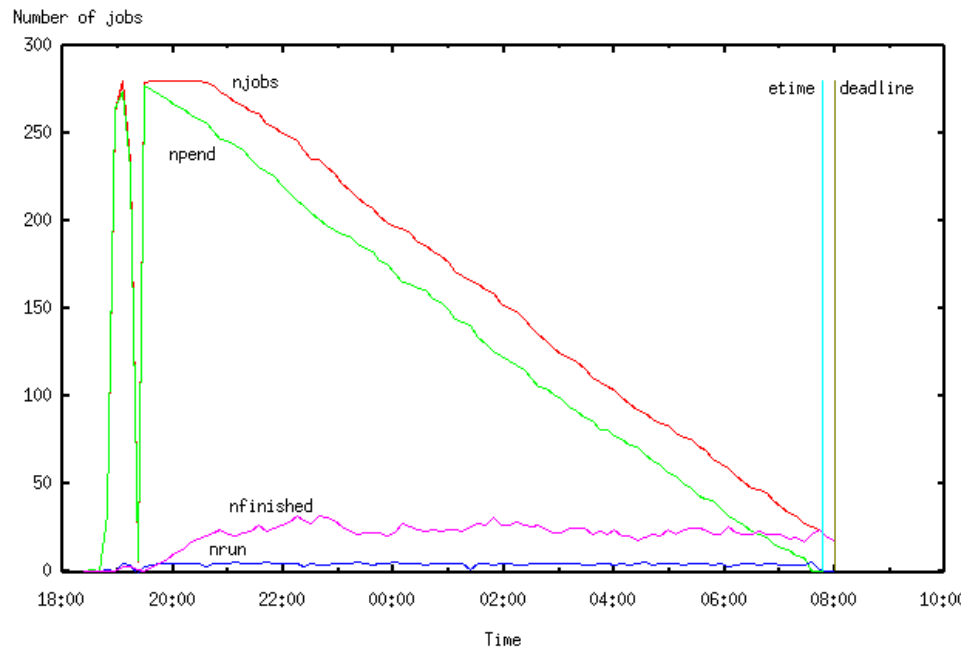
GOAL: VELOCITY 8
ACTIVE WINDOW: (8:00-18:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: DEADLINE
ACTIVE WINDOW: (18:00-8:00)
STATUS: Active:On time
ESTIMATED FINISH TIME: (Thu Jul 10 07:53)
OPTIMUM NUMBER OF RUNNING JOBS: 2
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 280   | 278  | 2   | 0     | 0     | 0      |

The following illustrates the progress of the deadline SLA Qualicum running 280 jobs overnight with random runtimes until the morning deadline. As with the simple deadline goal example, when the finished job curve (nfinished) meets the total number of jobs curve (njobs) the deadline is met with all jobs completed ahead of the configured deadline.

## Understanding Service Class Behavior



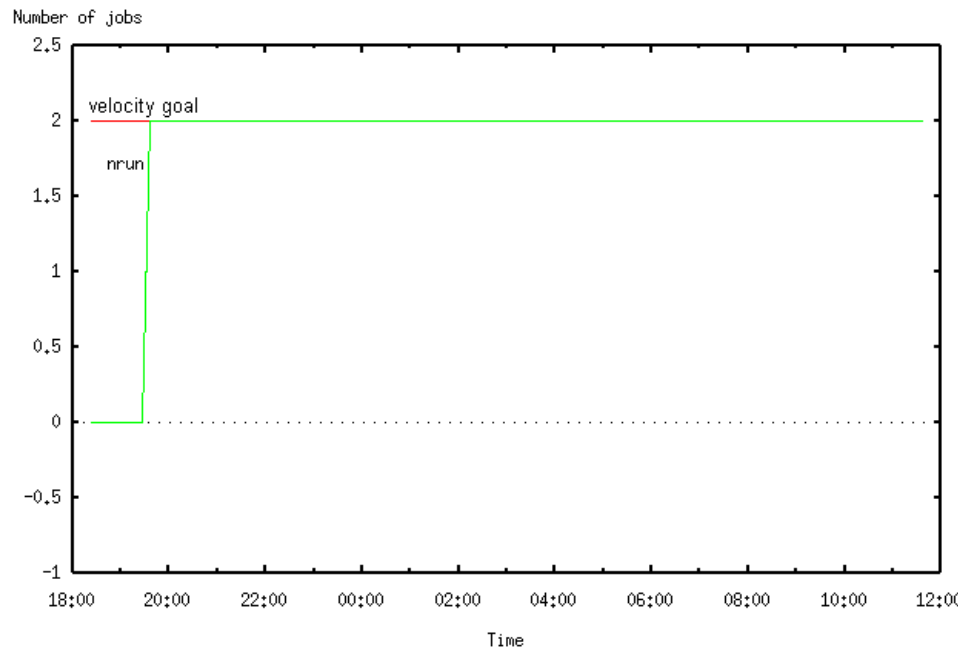
- ◆ Comox has a velocity goal of 2 concurrently running jobs that is always active:

```
bsla Comox
SERVICE CLASS NAME: Comox
PRIORITY: 20
```

```
GOAL: VELOCITY 2
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 2.00 JOBS/CLEAN_PERIOD
```

| NJOBS | PEND | RUN | SSUSP | USUSP | FINISH |
|-------|------|-----|-------|-------|--------|
| 100   | 98   | 2   | 0     | 0     | 0      |

The following illustrates the progress of the velocity SLA Comox running 100 jobs with random runtimes over a 14 hour period.



## When an SLA is missing its goal

- 1 Use the `CONTROL_ACTION` parameter in your service class to configure an action to be run if the SLA goal is delayed for a specified number of minutes.

### `CONTROL_ACTION` (lsb.serviceclasses)

```
CONTROL_ACTION=VIOLATION_PERIOD[minutes] CMD [action]
```

If the SLA goal is delayed for longer than `VIOLATION_PERIOD`, the action specified by `CMD` is invoked. The violation period is reset and the action runs again if the SLA is still active when the violation period expires again. If the SLA has multiple active goals that are in violation, the action is run for each of them.

### Example

```
CONTROL_ACTION=VIOLATION_PERIOD[10] CMD [echo `date`: SLA is in violation >> ! /tmp/sla_violation.log]
```

## Preemption and SLA policies

SLA jobs cannot be preempted. You should avoid running jobs belonging to an SLA in low priority queues.

## Chunk jobs and SLA policies

SLA jobs will not get chunked. You should avoid submitting SLA jobs to a chunk job queue.

## SLA statistics files

Each active SLA goal generates a statistics file for monitoring and analyzing the system. When the goal becomes inactive the file is no longer updated. The files are created in the `LSB_SHAREDIR/cluster_name/logdir/SLA` directory. Each file name consists of the name of the service class and the goal type.

For example the file named `Quadra.deadline` is created for the deadline goal of the service class name `Quadra`. The following file named `Tofino.velocity` refers to a velocity goal of the service class named `Tofino`:

```
cat Tofino.velocity
service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9 15:7:34 1063782454 2 0 0 0 0
17/9 15:8:34 1063782514 2 0 0 0 0
17/9 15:9:34 1063782574 2 0 0 0 0
service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9 15:10:10 1063782610 2 0 0 0 0
```

## Job groups and SLA scheduling

Job groups provide a method for assigning arbitrary labels to groups of jobs. Typically, job groups represent a project hierarchy. You can use `-g` with `-sla` at job submission to attach all jobs in a job group to a service class and have them scheduled as SLA jobs and subject to the scheduling policy of the SLA. Within the job group, resources are allocated to jobs on a fairshare basis.

All jobs submitted to a group under an SLA automatically belong to the SLA itself. You cannot modify a job group of a job that is attached to an SLA.

A job group hierarchy can belong to only one SLA.

It is not possible to have some jobs in a job group not part of the service class. Multiple job groups can be created under the same SLA. You can submit additional jobs to the job group without specifying the service class name again.

If the specified job group does not exist, it is created and attached to the SLA.

You can also use `-sla` to specify a service class when you create a job group with `bgadd`.

## View job groups attached to an SLA (bjgroup)

1 Run `bjgroup` to display job groups attached to a service class:

```
bjgroup
GROUP_NAME NJOBS PEND RUN SSUSP USUSP FINISH SLA JLIMIT OWNER
/fund1_grp 5 4 0 1 0 0 Venezia 1/5 user1
/fund2_grp 11 2 5 0 0 4 Venezia 5/5 user1
/bond_grp 2 2 0 0 0 0 Venezia 0/- user2
/risk_grp 2 1 1 0 0 0 () 1/- user2
/admi_grp 4 4 0 0 0 0 () 0/- user2
```

`bjgroup` displays the name of the service class that the job group is attached to with `bgadd -sla service_class_name`. If the job group is not attached to any service class, empty parentheses ( ) are displayed in the SLA name column.

## EGO-enabled SLA scheduling

By default, all host management for scheduling SLA jobs is handled by LSF. Under *EGO-enabled SLA scheduling*, LSF uses EGO resource allocation facilities to get the hosts it needs to run SLA jobs. Host allocation is the responsibility of EGO, while job management remains managed by LSF.

EGO-enabled SLA scheduling is a new scheduling paradigm that replaces other existing LSF scheduling policies. It effectively separates workload management from resource management. Because it takes advantage of the dynamic host feature, it allows fairsharing of compute host resources among multiple SLAs.

Hosts are assigned to a specific SLA and owned by it. Resources can be reclaimed and reallocated from and to consumers based on EGO resource policies. Each SLA defines one EGO consumer. Consumers can share the resources from one EGO resource group.

Attaching SLA scheduling service classes to EGO consumers provides:

- ◆ Two levels of fairshare:
  - ❖ Resource fairshare between service classes using EGO policies that determine how many hosts are to be assigned to a given SLA. This is similar to queue-level fairshare in LSF.
  - ❖ User-based fairshare is supported by defining the fairshare tree in `lsb.users`. The `USER_GROUP` parameter in `lsb.serviceclasses` controls access to the SLA by determining which user's job should run on the allocated hosts.
- ◆ Simplified scheduling policies—LSF focuses on job scheduling, the resource scheduling and distribution are managed by EGO
- ◆ Lending and borrowing of cluster resources among multiple consumers and projects, including immediate reclaim by resource owners

### Default EGO-enabled SLA behavior

With EGO-enabled SLA scheduling configured:

- ◆ Multiple LSF consumers are supported.
- ◆ Each LSF consumer requests exclusive host allocation.
- ◆ Hosts allocated to LSF are dedicated to LSF jobs; they cannot be shared with other consumers. You can configure your resource allocation plan in EGO to share hosts among consumers.
- ◆ By default, LSF schedule jobs based on the default LSF MXJ. Use `MBD_USE_EGO_MXJ` in `lsb.params` to configure slot-based allocation.
- ◆ LSF can only schedule jobs on hosts allocated from EGO, and the allocation can change with workload and demand. EGO-enabled SLA facilitates dynamic resource sharing and scheduling based on resource ownership.
- ◆ Resource requirements can be specified in the service class configuration to get specific resource.
- ◆ As with existing SLA scheduling in general, preemption and chunk job scheduling is not supported.

## Key concepts

### All hosts are dynamic for workload scheduling

`ENABLE_DEFAULT_EGO_SLA` in `lsb.params` is required to turn on EGO-enabled SLA scheduling.

The host resources assigned to LSF are subject to EGO resource allocation policies. Hosts are given to an SLA based on EGO allocation decisions; LSF decides which jobs and how many from each SLA will run.

All LSF compute host resource management is delegated to Platform EGO, and all LSF hosts are under EGO control. When all jobs running in the default SLA finish, all allocated hosts are released to EGO after the default idle timeout of 120 seconds (configurable by `MAX_HOST_IDLE_TIME` in `lsb.serviceclasses`).

Hosts allocated to a specific EGO-enabled SLA cannot be used by jobs belonging to other SLAs. Idle hosts are released back to EGO.

To ensure that all host resources are allocated to your jobs exclusively through EGO, you should not have host names configured statically in batch configuration files (`lsb.hosts`, `lsb.queues`, etc.). As a scheduling feature, all host resources are allocated dynamically to LSF workload by EGO. They flow in and out of LSF so that the number of hosts available to EGO-enabled SLAs grows and shrinks with demand, as represented by the velocity goals of the SLAs, and the number of pending jobs in the system.

LSF still maintains the visibility of all hosts —they are still displayed by `bhosts`, but they are not usable for scheduling unless they have been allocated to LSF by EGO.

### Special LSF host group for EGO-enabled SLA

When hosts are allocated to an EGO-enabled SLA, they are automatically added to an LSF host group created by the SLA. The name of the host group is `_sla_sla_name`, where `sla_name` is the name of the EGO-enabled SLA defined in `lsb.serviceclasses` or in `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`. One of the hosts in the host group will have the name `_virtual`.

When the host is released to EGO, the entry is removed from the host group. `bmgroup` displays the hosts allocated by EGO to the host group created by the SLA.

By default, all hosts that are not allocated to LSF by EGO are closed with reason `Closed by EGO`. Hosts are opened only when they are assigned by EGO. When LSF returns them to EGO it closes them again.

Run `bmgroup` to display the hosts dynamically allocated by EGO to the host group created by the SLA.

### EGO consumers mapped to SLAs

Multiple EGO consumers are mapped to different service classes in `lsb.serviceclasses`. This enables LSF to allocate job resources to SLAs with different workload priorities and demand requirements (for example, projects with different priorities, or daytime versus nighttime workload). At the same time, EGO-enabled SLAs ensure hosts are shared appropriately and reclaimed through EGO by the consumers that own them.

Jobs are submitted to the appropriate SLA, and EGO allocates hosts for the job according to the ownership and distribution policies of the consumers associated with the SLA. All other LSF jobs that do not explicitly define an SLA run under the SLA defined by `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`.

**SLA goal behavior** Only velocity goals are supported in EGO-enabled SLAs. Deadline goals and throughput goals are not supported. For EGO-enabled SLA, the configured velocity value is considered to be a *minimum* number of jobs that should be in run state from the SLA.

This is different from a regular SLA, where once the velocity is reached, no more jobs are dispatched by the SLA. Under EGO-enabled SLA, if pending jobs exist, the SLA will try to run them all. If the goal becomes inactive before all jobs are dispatched, LSF continues to dispatch the remaining pending jobs until all the jobs under the SLA are finished, then returns the host resources to EGO after the idle timeout.

---

**TIP:** To ensure a steady flow of jobs through the system for your velocity goals, you should set the ownership for the consumer representing the SLA to a value equal to the desired velocity.

---

If the status of the SLA goal is inactive when the job is submitted, the job will remain pending and LSF will not attempt to get hosts from EGO. This is different from regular SLAs, where jobs submitted to an inactive SLA are dispatched as if they were not submitted to any SLA.

**How LSF gets host resources from EGO** Each SLA computes its goal and how many hosts it needs for LSF to request from EGO. Hosts to be allocated to SLAs are configured in EGO resource groups. Allocated hosts are added to the LSF host group automatically created for the SLA, which can then start its jobs.

By default, EGO allocates an entire host to LSF, which uses its own MXJ definition to determine how many slots are available on the host. Configure `MBD_USE_EGO_MXJ` in `lsb.params` to configure slot-based allocation.

**How host resources return to EGO** Hosts are returned from LSF to EGO two ways:

**Voluntarily** All allocated hosts are released to EGO if no jobs are running on the host after the default idle timeout of 120 seconds (configurable by `MAX_HOST_IDLE_TIME` in `lsb.serviceclasses`). If you configure `MBD_USE_EGO_MXJ` in `lsb.params` to configure slot-based allocation, LSF returns idle slots to EGO.

**Reclaim** EGO can reclaim hosts allocated to LSF service classes at any time and reallocates them to other consumers. When LSF receives a reclaim request from EGO, it kills and requeues the jobs to pending state on the reclaimed host. If you configure `MBD_USE_EGO_MXJ` in `lsb.params` to configure slot-based allocation, EGO reclaims the allocated slots.

## Configure EGO-enabled SLA scheduling

**lsb.params** Minimal configuration enables SLA on EGO. Advanced configuration in controls LSF behavior on EGO.

**lsb.serviceclasses** Service class configuration maps an SLA to a specific consumer in EGO. Advanced configuration controls host idle period and SLA evaluation period.

## Turn on basic EGO-enabled SLA scheduling

**Prerequisites:** To use EGO-enabled SLA scheduling, all hosts that the SLA will use must be dynamically allocated by EGO.

All hosts that the SLA will use must be dynamically allocated by EGO:

- ◆ Edit `lsb.hosts` to remove static hosts and host groups containing the hosts you want EGO to allocate.

---

**IMPORTANT:** The `lsb.hosts` file must contain a "default" host line. For example:

```
Begin Host
HOST_NAME MXJ r1m pg ls tmp DISPATCH_WINDOW # Keywords
default ! () () () () () # Example
End Host
```

---

- ◆ Edit `lsb.queues` to remove static hosts and host groups containing the hosts you want EGO to allocate from the queues you want to submit EGO-enabled SLA jobs to.
- ◆ Queues and host partitions must be configured with `HOSTS = all`.

---

`ENABLE_DEFAULT_EGO_SLA` in `lsb.params` configures a global SLA for all jobs in the system. Jobs submitted without `-sla` are attached to the default service class. All LSF scheduling policies apply.

- 
- 1 Log on to the LSF master host as the LSF administrator (`lsfadmin`).
  - 2 Let all LSF workload run to completion.
  - 3 Shut down the LSF cluster.
  - 4 Edit `lsb.params` and set `ENABLE_DEFAULT_EGO_SLA` to the name of the the default SLA.

```
ENABLE_DEFAULT_EGO_SLA=Venezia
```

The value of `ENABLE_DEFAULT_EGO_SLA` is the name of a valid EGO consumer. If a service class containing a consumer with the same name does not exist in `lsb.servieclasses`, LSF creates a dynamic service class with the specified consumer name, a velocity of 1, and a time window that is always open.

- 5 Run `badmin reconfig` to reconfigure `mbatchd`.
- 

## Configure service classes for EGO-enabled SLA

**Prerequisites:** EGO-enabled SLA scheduling must be turned on with `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`.



---

Configure service classes for multiple SLAs that map to different EGO consumers.

---

- 1 Log in as the LSF administrator on any host in the cluster.
- 2 Edit `lsb.serviceclasses` to add the service class definition for EGO-enabled SLA.

```
Begin ServiceClass
NAME = Venezia
CONSUMER = Venezia
PRIORITY = 10
GOALS = [VELOCITY 10 timeWindow (8:00-18:00)]
MAX_HOST_IDLE_TIME = 10
DESCRIPTION = Daytime SLA
End ServiceClass
```

You can copy another service class definition from this file as a starting point; remember to change the `NAME` and `CONSUMER` of the copied profile.

Configure the additional parameters for an EGO-enabled SLA:

- ◆ `CONSUMER=ego_consumer_name`—enables an SLA to get resources from EGO. Specify the name of the EGO consumer from which hosts are allocated to the SLA. This parameter is mandatory for EGO-enabled SLA.

---

**TIP:** You should make the service class name the same as the consumer name.

---

- 3 Save the changes to `lsb.serviceclasses`.
  - 4 Run `badmin reconfig` to reconfigure `mbatchd`.
- 

## Configure EGO consumers and resource groups

**Prerequisites:** You must be logged on as an EGO cluster administrator and you should have already added hosts to your cluster. To create consumers, you must either be a cluster administrator or a consumer administrator for the branch on which you are creating the consumer.

---

- 1 Log on to the Platform Management Console.
- 2 Build the consumer tree for SLA jobs.  
See “Create a consumer” in *Administering and Using Platform EGO* for detailed steps.

The Platform EGO consumer tree organizes consumers into a structure that makes it easy to apply resource plans. The plan cannot be defined without the consumer tree. The tree defines organizational relationships among consumers. The plan defines the resource allocation.

- 3 Configure resource groups for SLA jobs.  
See “Create a resource group by host names” or “Create a resource group by resource requirement” in *Administering and Using Platform EGO* for detailed steps.

Create resource groups from the Platform Management Console to ensure your consumers have the appropriate group of hosts available to them according to the requirements of the workload.

You can create resource groups by resource requirement (dynamic—specific member hosts not specified) or by host name (static—only hosts currently in the cluster are members; if you add new hosts to the cluster, you must manually add them to a static resource group.)

---

**NOTE:** Unlike host groups in LSF, hosts should not overlap between resource groups. Resource groups determine the distribution of resources in your EGO resource plan. Having overlaps causes the hosts to be double counted in the resource plan, resulting in overcommitment of hosts to some consumers.

---

**4 Associate resource groups to consumers.**

So that Platform EGO allocates resources how and when you want them, create a resource plan to distribute resources from each resource group to each consumer.

See “Create or modify a resource plan” in *Administering and Using Platform EGO* for detailed steps.

**5 Configure a default SLA in or service classes in `lsb.serviceclasses` to use these consumers.**

---

## Submit jobs to an EGO-enabled SLA

---

**1 Run `bsub -sla` to submit jobs to an EGO-enabled SLA.**

You can use `-g` with `-sla` to attach all jobs in a job group to a service class and have them scheduled as SLA jobs.

Jobs submitted without `-sla` are attached to the SLA defined in `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`.

---

## View jobs running in an EGO-enabled SLA (`bsla`)

The `bsla` command displays additional keywords for EGO-enabled SLA.

---

**1 Run `bsla` to display jobs running in a service class:**

```
bsla 'Kyuuquot'
SERVICE CLASS NAME: 'Kyuuquot'
 -- Daytime/Nighttime SLA
PRIORITY: 23
CONSUMER: 'Kyuuquot'
EGO_RES_REQ: any host
MAX_HOST_IDLE_TIME: 120
USER GROUP: user1 user2
GOAL: VELOCITY 8
ACTIVE WINDOW: (9:00-17:30)
STATUS: Active:On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
 NJOBS PEND RUN SSUSP USUSP FINISH
```

```
4 0 4 0 0 0
```

If the SLA is under reclaim, `bsla` displays the following keywords:

- ◆ `NUM_RECALLED_HOSTS`
- ◆ `RECALLED_HOSTS_TIMEOUT`

If a default system service class is configured with `ENABLE_DEFAULT_EGO_SLA` in `lsb.params` but no other service classes are explicitly configured in `lsb.serviceclasses`, `bsla` only displays information for the default SLA.

## Changing existing configurations to use EGO-enabled SLA scheduling

When the EGO-enabled SLA is configured, you no longer need static hosts or host groups, or host partitions.

Changing your configuration is not required for EGO-enabled SLA scheduling to work. However, if you explicitly configure hosts or host groups in your batch configuration there is no guarantee that EGO will allocate those hosts to your jobs. If your business goals require guaranteed availability of resources, you should change your configuration so that all hosts are allocated from EGO.

### Static hosts, host groups, and host partitions

A typical LSF configuration divides hosts according to application requirements, job resource requirements, project ownership, and fairshare. Administrators usually configure host groups or host partitions to accomplish this.

Under EGO-enabled SLA scheduling, you should:

- ◆ Replace host groups with EGO resource groups by removing static host and host group configurations from `lsb.hosts` and `lsb.queues`.

**TIP:** You should remove any overlap among your host groups so that you have no host overlap in the EGO resource groups.

- ◆ Replace host partitions with EGO consumers.
- ◆ Move resource requirement parameters from the queue to the EGO-enabled service class.
- ◆ Submit jobs attached to an SLA so that the SLA consumer gets its host resources from the EGO resource group.

### SLA priority

You may have both high priority and low priority jobs where you want to preempt the low priority jobs when high priority jobs are pending. EGO-enabled SLA uses EGO resource ownership to enable SLA priority. You can configure two SLAs with two different EGO consumers and different SLA priorities. The consumer associated with the higher priority SLA owns the resources and the consumer associated with the lower priority SLA borrows resources from high priority consumer. When a low priority job is running, and the high priority consumer has pending jobs, EGO reclaims resources from the low priority SLA and assigns them to the high priority SLA, allowing the pending job from the high priority SLA to run.

EGO resource reclaim is not exactly the same as LSF preemption. Preempted jobs in LSF are suspended, while jobs reclaimed from the EGO-enabled SLA are requeued.

LSF preemption also supports queues that can be both preemptable and preemptive. Under the EGO reclaim mechanism, resource ownership guarantees that some jobs cannot be reclaimed.

### Job and queue priority

Queue priority is respected by EGO-enabled SLA scheduling. You can also configure priority among SLAs. Configure the consumer rank in your EGO consumer tree to reflect the priorities in your SLAs. See *Administering and Using Platform EGO* for information about setting the priority of consumers.

### Example: change host groups to EGO resource groups

Host groups are no longer needed with EGO-enabled SLA scheduling. You can configure resource groups two ways:

- ◆ By resource requirement, using specific resources, like memory, processor, operating system, etc,
- ◆ By host name, selecting specific member hosts

### Configure specific resources for job scheduling

**Prerequisites:** EGO-enabled SLA scheduling must be turned on with `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`. The Platform EGO consumers, resource groups, and resource plan that you want to use for the EGO-enabled SLA configuration must already be set up.

All hosts that the SLA will use must be dynamically allocated by EGO:

- ◆ Edit `lsb.hosts` to remove static hosts and host groups containing the hosts you want EGO to allocate.

---

**IMPORTANT:** The `lsb.hosts` file must contain a "default" host line. For example:

```
Begin Host
HOST_NAME MXJ r1m pg ls tmp DISPATCH_WINDOW # Keywords
default ! () () () () () # Example
End Host
```

- ◆ Edit `lsb.queues` to remove static hosts and host groups containing the hosts you want EGO to allocate from the queues you want to submit EGO-enabled SLA jobs to.
- ◆ Queues and host partitions must be configured with `HOSTS = all`.

---

Complete the following steps to configure an SLA to schedule jobs based on specific resource requirements.

- 1 Log on to the LSF master host as the LSF administrator (`lsfadmin`).
- 2 Log on to the Platform Management Console:
  - a Create an EGO resource group with specific resource requirements.
  - b Define an EGO consumer for the SLA (for example, `LSF_SLA`).
  - c Define the EGO resource plan for the resource group associated with the consumer `LSF_SLA`.
- 3 Edit `lsf.shared` and define shared resources for your different host requirements.

For example:

```
special Boolean () () (resource locked for group 1)
bigmem Boolean () () (machine with big memory)
```

- 4 Edit `lsb.queues` and remove any existing resource requirements (`RES_REQ`) that specify these shared resources.

- 5 Edit `lsb.serviceclasses` and define an SLA that uses `LSF_SLA` as the consumer, and requires the shared resources.

```
Begin ServiceClass
NAME=Uclulet
PRIORITY=20
GOALS=[Velocity timeWindow ()]
CONSUMER=LSF_SLA
EGO_RES_REQ="select(special && bigmem)"
End ServiceClass
```

- 6 Edit `lsf.cluster.cluster_name` and define hosts with the shared resources.

```
hostA ! RHEL_64 0 (special bigmem)
hostB ! RHEL_64 0 (special)
hostC ! RHEL_64 0 (bigmem)
```

- 7 Run `badadmin reconfig` to reconfigure `mbatchd`.

With this configuration, EGO allocates host `hostA` to the service class `Uclulet`. All jobs submitted to service class `Uclulet` can only be scheduled to host `hostA`.

## Configure specific hosts for job scheduling

**Prerequisites:** EGO-enabled SLA scheduling must be turned on with `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`. The Platform EGO consumers, resource groups, and resource plan that you want to use for the EGO-enabled SLA configuration must already be set up.

All hosts that the SLA will use must be dynamically allocated by EGO:

- ◆ Edit `lsb.hosts` to remove static hosts and host groups containing the hosts you want EGO to allocate.

**IMPORTANT:** The `lsb.hosts` file must contain a "default" host line. For example:

```
Begin Host
HOST_NAME MXJ r1m pg 1s tmp DISPATCH_WINDOW # Keywords
default ! () () () () () # Example
End Host
```

- ◆ Edit `lsb.queues` to remove static hosts and host groups containing the hosts you want EGO to allocate from the queues you want to submit EGO-enabled SLA jobs to.
- ◆ Queues and host partitions must be configured with `HOSTS = all`.

---

Complete the following steps to configure an SLA to schedule jobs using specific hosts.

- 1 Log on to the LSF master host as the LSF administrator (`lsfadmin`).
  - 2 Let all LSF workload run to completion.
  - 3 Shut down the LSF cluster.
  - 4 Edit `lsb.queues` and remove any hosts and host groups from the queues you want to submit EGO-enabled SLA jobs to.
  - 5 Log on to the Platform Management Console:
    - a Create an EGO resource group that contains specific hosts (for example, `hostA` and `hostB`).
    - b Define an EGO consumer for the SLA (for example, `LSF_SLA`).
    - c Define the EGO resource plan for the resource group associated with the consumer `LSF_SLA`.
  - 6 Edit `lsb.serviceclasses` and define an SLA that uses `LSF_SLA` as the consumer.
  - 7 Run `badmin reconfig` to reconfigure `mbatchd`.
- 

With this configuration, EGO allocates hosts `hostA` and `hostB` to the SLA. All jobs submitted to SLA `Uc1ulet` can only be scheduled to hosts `hostA` and `hostB`.

## Example: change host partitions to EGO consumers

**Prerequisites:** EGO-enabled SLA scheduling must be turned on with `ENABLE_DEFAULT_EGO_SLA` in `lsb.params`. The Platform EGO consumers, resource groups, and resource plan that you want to use for the EGO-enabled SLA configuration must already be set up.

All hosts that the SLA will use must be dynamically allocated by EGO:

- ◆ Edit `lsb.hosts` to remove static hosts and host groups containing the hosts you want EGO to allocate.

---

**IMPORTANT:** The `lsb.hosts` file must contain a "default" host line. For example:

```
Begin Host
HOST_NAME MXJ r1m pg ls tmp DISPATCH_WINDOW # Keywords
default ! () () () () () # Example
End Host
```

---

- ◆ Edit `lsb.queues` to remove static hosts and host groups containing the hosts you want EGO to allocate from the queues you want to submit EGO-enabled SLA jobs to.
- ◆ Queues and host partitions must be configured with `HOSTS = all`.

LSF host partitions are typically used to implement user-based fairshare policies. Complete the following steps to allow the EGO-enabled SLA to allocate hosts to specific users.

- 1 Log on to the LSF master host as the cluster administrator.
- 2 Log on to the Platform Management Console:
  - a Define an EGO resource group that contains the selected hosts.
  - b Define an EGO consumer that is associated with this resource group.
- 3 Edit `lsb.users` and define a user group that configures the fairshare policy.
- 4 Edit `lsb.serviceclasses` and define a service class that uses the consumer and user group.
- 5 Run `badmin reconfig` to reconfigure `mbatchd`.

With this configuration, when the job is submitted to the defined SLA, the job is scheduled according to the user-based fairshare policy within the selected hosts.

## Advanced configuration

### `lsb.serviceclasses`

- ◆ `EGO_RES_REQ=res_req`—EGO resource requirement that specifies the characteristics of the hosts that EGO will assign to the SLA. Must be a valid EGO resource requirement. The EGO resource requirement string supports the `select` section, but the format is different from LSF resource requirements. For example:
 

```
EGO_RES_REQ=select(linux && maxmem > 100)
```
- ◆ `MAX_HOST_IDLE_TIME=seconds`—number of seconds that the SLA will hold on to hosts that have no jobs running before LSF releases them to EGO. Each SLA can configure a different idle time. The default is 120 seconds. Do not set this parameter to a small value, or LSF may release hosts too quickly.

### `lsb.params`

- ◆ `DEFAULT_SLA_VELOCITY`—the number of slots that the SLA should request for parallel jobs running in the SLA. By default, an EGO-enabled SLA requests slots from EGO based on the number of jobs the SLA needs to run, not the number of slots required by the job. If the jobs themselves require more than one slot, they will remain pending. To avoid this for parallel jobs, set `DEFAULT_SLA_VELOCITY` to the total number of slots that are expected to be used by parallel jobs.
- ◆ `MBD_EGO_CONNECT_TIMEOUT`—timeout parameter for network I/O with EGO `vemkd`. The default is 3 seconds.
- ◆ `MBD_EGO_READ_TIMEOUT`—timeout parameter for network I/O with EGO `vemkd`. The default is 3 seconds.
- ◆ `MBD_EGO_TIME2LIVE`—how long EGO should keep information about host allocations in case `mbatchd` restarts. The default is 1440 minutes (24 hours).

- ◆ **SLA\_TIMER**—controls how often each service class is evaluated and a network message is sent to EGO communicating host demand. The default is 10 seconds.

## LSF MXJ and EGO slots

In LSF, you configure the maximum number of job slots (MXJ) in `lsb.hosts`. By default, the MXJ equals to the number of processors on the host. LSF schedules jobs on that host based on the MXJ.

By default, when EGO-enabled SLA scheduling is configured, EGO allocates an entire host to LSF, which uses its own MXJ definition to determine how many slots are available on the host. LSF gets its host allocation from EGO, and runs as many jobs as the LSF configured MXJ for that host dictates.

To allow partial sharing of hosts (for example, a large SMP computer) among different consumers or workload managers, use `MBD_USE_EGO_MXJ` in `lsb.params`. This forces LSF to use the job slot maximum configured in the EGO consumer. When `MBD_USE_EGO_MXJ` is set, LSF schedules jobs based on the number of slots allocated from EGO.

When `MBD_USE_EGO_MXJ` is set, the number of maximum jobs is set to the number of slots EGO allocates to LSF. For example, if `hostA` has 4 processors, but EGO allocates 2 slots to an EGO-enabled SLA consumer. LSF can schedule a maximum of 2 jobs from that SLA on `hostA`.

LSF cannot release idle slots until while jobs are running on the host. For example, if EGO allocates 3 slots on a host, and one job finishes before the other two, the idle slot is not available to other SLAs until all jobs finish on the host.

## Limitations of EGO-enabled SLA scheduling

- Parallel jobs** Resource allocation is based on the number of jobs, not the slots required by the job. EGO-enabled SLA requests resource based on velocity and the number of pending jobs. If a parallel job requires multiple processors, the SLA may request fewer processors than the requirement, which causes the job to remain pending. To avoid this, you can configure larger velocity in the SLA.
- Multicluster** Resource export under the lease model is not guaranteed. With EGO-enabled SLA scheduling, all resources are dynamic, so the exported hosts may not be allocated to LSF.
- Advance reservations** EGO-enabled SLA does not support advance reservations. Advanced reservations need to reserve resources for a specified time window, which is not currently supported in EGO.
- Job-level resource requirements (bsub -R)** LSF takes the resource requirement into consideration for scheduling, but if the resource request does not match the resource requirement specified in the service class, the host allocated by EGO cannot match the specified resource requirement, and the job remains pending. LSF treats the allocated host as idle and returns it to EGO. The pending job causes another request to be sent to EGO, which allocates another host, which may or may not satisfy the resource requirement.  
Use `EGO_RES_REQ=res_req` in the service class configuration to specify all job resource requirements.



**Job-level host preference (bsub -m)** Specific job-level host requests are similar to `bsub -R` (essentially the same as `bsub -R "select host_name"`). The specified host is not guaranteed to be allocated by EGO. The job remains pending until the specified host actually allocated.

Use `EGO_RES_REQ=res_req` in the service class configuration to specify all job resource requirements.

### For more information

See “Managing Resource Distribution” in *Administering and Using Platform EGO* for detailed information about creating and managing Platform EGO consumers, resource groups, and resource plans.



## Job Scheduling and Dispatch

- ◆ [Resource Allocation Limits](#) on page 369
- ◆ [Reserving Resources](#) on page 387
- ◆ [Advance Reservation](#) on page 403
- ◆ [Dispatch and Run Windows](#) on page 419
- ◆ [Job Dependencies](#) on page 423
- ◆ [Job Priorities](#) on page 429
- ◆ [Job Requeue and Job Rerun](#) on page 441
- ◆ [Job Checkpoint, Restart, and Migration](#) on page 447
- ◆ [Chunk Job Dispatch](#) on page 453
- ◆ [Job Arrays](#) on page 459
- ◆ [Running Parallel Jobs](#) on page 469
- ◆ [Submitting Jobs Using JSDL](#) on page 495



## Working with Application Profiles

*Application profiles* improve the management of applications by separating scheduling policies (preemption, fairshare, etc.) from application-level requirements, such as pre-execution and post-execution commands, resource limits, or job controls, job chunking, etc.

### Contents

- ◆ [Manage application profiles](#) on page 357
- ◆ [View application profile information](#) on page 360
- ◆ [Use application profiles](#) on page 359
- ◆ [How application profiles interact with queue and job parameters](#) on page 362

### Manage application profiles

#### About application profiles

Use application profiles to map common execution requirements to application-specific job containers. For example, you can define different job types according to the properties of the applications that you use; your FLUENT jobs can have different execution requirements from your CATIA jobs, but they can all be submitted to the same queue.

The following application profile defines the execution requirements for the FLUENT application:

```
Begin Application
NAME = fluent
DESCRIPTION = FLUENT Version 6.2
CPULIMIT = 180/hostA # 3 hours of host hostA
FILELIMIT = 20000
DATALIMIT = 20000 # jobs data segment limit
CORELIMIT = 20000
PROCLIMIT = 5 # job processor limit
PRE_EXEC = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
```

## Manage application profiles

```
REQUEUE_EXIT_VALUES = 55 34 78
```

```
End Application
```

See the `lsb.applications` template file for additional application profile examples.

## Add or remove application profiles

### Add an application profile

- 1 Log in as the LSF administrator on any host in the cluster.
- 2 Edit `lsb.applications` to add the new application profile definition.  
You can copy another application profile definition from this file as a starting point; remember to change the `NAME` of the copied profile.
- 3 Save the changes to `lsb.applications`.
- 4 Run `badadmin reconfig` to reconfigure `mbatchd`.

Adding an application profile does not affect pending or running jobs.

### Remove an application profile

**Prerequisites:** Before removing an application profile, make sure there are no pending jobs associated with the application profile.

If there are jobs in the application profile, use `bmod -app` to move pending jobs to another application profile, then remove the application profile. Running jobs are not affected by removing the application profile associated with them,

**NOTE:** You cannot remove a default application profile.

- 1 Log in as the LSF administrator on any host in the cluster.
- 2 Run `bmod -app` to move all pending jobs into another application profile.  
If you leave pending jobs associated with an application profile that has been removed, they remain pending with the pending reason  
`Specified application profile does not exist`
- 3 Edit `lsb.applications` and remove or comment out the definition for the application profile you want to remove.
- 4 Save the changes to `lsb.applications`.
- 5 Run `badadmin reconfig` to reconfigure `mbatchd`.

## Define a default application profile

Define a default application profile that is used when a job is submitted without specifying an application profile,

- 1 Log in as the LSF administrator on any host in the cluster.
- 2 Set `DEFAULT_APPLICATION` in `lsb.params` to the name of the default application profile.  
`DEFAULT_APPLICATION=catia`

- 3 Save the changes to `lsb.params`.
  - 4 Run `badmin reconfig` to reconfigure `mbatchd`.  
Adding an application profile does not affect pending or running jobs.
- 

## Use application profiles

### Submit jobs to application profiles

Use the `-app` option of `bsub` to specify an application profile for the job.

---

- 1 Run `bsub -app` to submit jobs to an application profile.  
`bsub -app fluent -q overnight myjob`  
LSF rejects the job if the specified application profile does not exist.
- 

### Modify the application profile associated with a job

**Prerequisites:** You can only modify the application profile for pending jobs.

---

- 1 Run `bmod -app application_profile_name` to modify the application profile of the job.

The `-appn` option dissociates the specified job from its application profile. If the application profile does not exist, the job is not modified

---

```
bmod -app fluent 2308
```

Associates job 2308 with the application profile `fluent`.

```
bmod -appn 2308
```

Dissociates job 2308 from the application profile `fluent`.

### Control jobs associated with application profiles

`bstop`, `bresume`, and `bkill` operate on jobs associated with the specified application profile. You must specify an existing application profile. If `job_ID` or `0` is not specified, only the most recently submitted qualifying job is operated on.

---

- 1 Run `bstop -app` to suspend jobs in an application profile.  
`bstop -app fluent 2280`  
Suspends job 2280 associated with the application profile `fluent`.  
`bstop -app fluent 0`  
Suspends all jobs associated with the application profile `fluent`.
- 2 Run `bresume -app` to resume jobs in an application profile.  
`bresume -app fluent 2280`  
Resumes job 2280 associated with the application profile `fluent`.
- 3 Run `bkill -app` to kill jobs in an application profile.  
`bkill -app fluent`

## View application profile information

Kills the most recently submitted job associated with the application profile `fluent` for the current user.

```
bkill -app fluent 0
```

Kills all jobs associated with the application profile `fluent` for the current user.

## View application profile information

| To view the...                                                             | Run...                                                     |
|----------------------------------------------------------------------------|------------------------------------------------------------|
| Available application profiles                                             | <code>bapp</code>                                          |
| Detailed application profile information                                   | <code>bapp -l</code>                                       |
| Jobs associated with an application profile                                | <code>bjobs -l -app <i>application_profile_name</i></code> |
| Accounting information for all jobs associated with an application profile | <code>bacct -l -app <i>application_profile_name</i></code> |

## View available application profiles

- 1 Run `bapp`. You can view a particular application profile or all profiles.

```
bapp
APPLICATION_NAME NJOBS PEND RUN SUSP
fluent 0 0 0 0
catia 0 0 0 0
```

A dash (-) in any entry means that the column does not apply to the row.

## View detailed application profile information

- 1 To see the complete configuration for each application profile, run `bapp -l`. `bapp -l` also gives current statistics about the jobs in a particular application profile, such as the total number of jobs in the profile, the number of jobs running, suspended, and so on.

Specify application profile names to see the properties of specific application profiles.

```
bapp -l fluent
```

```
APPLICATION NAME: fluent
```

```
-- Application definition for Fluent v2.0
```

```
STATISTICS:
```

| NJOBS | PEND | RUN | SSUSP | USUSP | RSV |
|-------|------|-----|-------|-------|-----|
| 0     | 0    | 0   | 0     | 0     | 0   |

```
PARAMETERS:
```

```
CPULIMIT
```

```
600.0 min of hostA
```

```
RUNLIMIT
```



```

200.0 min of hostA
PROCLIMIT
9
FILELIMIT DATALIMIT STACKLIMIT CORELIMIT MEMLIMIT SWAPLIMIT
PROCESSLIMIT THREADLIMIT
800 K 100 K 900 K 700 K 300 K 1000 K 400
500
RERUNNABLE: Y
CHUNK_JOB_SIZE: 5

```

---

## View jobs associated with application profiles

---

1 Run `bjobs -l -app application_profile_name`.

```
bjobs -l -app fluent
```

```

Job <1865>, User <user1>, Project <default>, Application <fluent>,
 Status <PSUSP>, Queue <normal>, Command <ls>
Tue Jun 6 11:52:05: Submitted from host <hostA> with hold, CWD
 </clusters/lsf7/work/cluster1/logdir>;

```

PENDING REASONS:

Job was suspended by LSF admin or root while pending;

SCHEDULING PARAMETERS:

|           | r15s     | r1m | r15m      | ut | pg | io | ls | it | tmp | swp | mem | tlu |
|-----------|----------|-----|-----------|----|----|----|----|----|-----|-----|-----|-----|
| loadSched | -        | -   | -         | -  | -  | -  | -  | -  | -   | -   | -   | -   |
| loadStop  | -        | -   | -         | -  | -  | -  | -  | -  | -   | -   | -   | -   |
|           | cpuspeed |     | bandwidth |    |    |    |    |    |     |     |     |     |
| loadSched | -        |     | -         |    |    |    |    |    |     |     |     |     |
| loadStop  | -        |     | -         |    |    |    |    |    |     |     |     |     |

A dash (-) in any entry means that the column does not apply to the row.

---

## Accounting information for all jobs associated with an application profile

---

1 Run `bacct -l -app application_profile_name`.

```
bacct -l -app fluent
```

Accounting information about jobs that are:

- submitted by users jchan,
  - accounted on all projects.
  - completed normally or exited
  - executed on all hosts.
  - submitted to all queues.
  - accounted on all service classes.
  - associated with application profiles: fluent
-

## How application profiles interact with queue and job parameters

```
Job <207>, User <user1>, Project <default>, Application <fluent>, Status <DONE>
 , Queue <normal>, Command <dir>
Wed May 31 16:52:42: Submitted from host <hostA>, CWD <${HOME}/src/mainline/lSBATCH
 /cmd>;
Wed May 31 16:52:48: Dispatched to 10 Hosts/Processors <10*hostA>
Wed May 31 16:52:48: Completed <done>.
Accounting information about this job:
 CPU_T WAIT TURNAROUND STATUS HOG_FACTOR MEM SWAP
 0.02 6 6 done 0.0035 2M 5M

...
SUMMARY: (time unit: second)
Total number of done jobs: 15 Total number of exited jobs: 4
Total CPU time consumed: 0.4 Average CPU time consumed: 0.0
Maximum CPU time of a job: 0.0 Minimum CPU time of a job: 0.0
Total wait time in queues: 5305.0
Average wait time in queue: 279.2
Maximum wait time in queue: 3577.0 Minimum wait time in queue: 2.0
Average turnaround time: 306 (seconds/job)
Maximum turnaround time: 3577 Minimum turnaround time: 5
Average hog factor of a job: 0.00 (cpu time / turnaround time)
Maximum hog factor of a job: 0.01 Minimum hog factor of a job: 0.00
Total throughput: 0.14 (jobs/hour) during 139.98 hours
Beginning time: May 31 16:52 Ending time: Jun 6 12:51
```

---

## How application profiles interact with queue and job parameters

Application profiles operate in conjunction with queue and job-level options. In general, you use application profile definitions to refine queue-level settings, or to exclude some jobs from queue-level parameters.

### Application profile settings that override queue settings

The following application profile parameters override the corresponding queue setting:

- ◆ JOB\_STARTER
- ◆ REQUEUE\_EXIT\_VALUES
- ◆ RESUME\_CONTROL—overrides queue JOB\_CONTROLS
- ◆ SUSPEND\_CONTROL—overrides queue JOB\_CONTROLS
- ◆ TERMINATE\_CONTROL—overrides queue JOB\_CONTROLS

## Application profile limits and queue limits

The following application profile limits override the corresponding queue-level soft limits:

- ◆ CORELIMIT
- ◆ CPULIMIT
- ◆ DATALIMIT
- ◆ FILELIMIT
- ◆ MEMLIMIT
- ◆ PROCESSLIMIT
- ◆ RUNLIMIT
- ◆ STACKLIMIT
- ◆ SWAPLIMIT
- ◆ STACKLIMIT
- ◆ THREADLIMIT

Job-level limits can override the application profile limits. The application profile limits cannot override queue-level hard limits.

### Processor limits

PROCLIMIT in an application profile specifies the maximum number of slots that can be allocated to a job. For parallel jobs, PROCLIMIT is the maximum number of processors that can be allocated to the job.

You can optionally specify the minimum and default number of processors. All limits must be positive integers greater than or equal to 1 that satisfy the following relationship:

$$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$$

Job-level processor limits (`bsub -n`) override application-level PROCLIMIT, which overrides queue-level PROCLIMIT. Job-level limits must fall within the maximum and minimum limits of the application profile and the queue.

### Absolute run limits

If you want the scheduler to treat any run limits as absolute, define `ABS_RUNLIMIT=Y` in `lsb.params` or in `lsb.applications` for the application profile associated with your job. When `ABS_RUNLIMIT=Y` is defined in `lsb.params` or in the application profile, the run time limit is not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit configured.

### Pre-execution

Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (`bsub -E`) override application-level pre-execution commands.

### Post-execution

When a job finishes, application-level post-execution commands run, followed by queue-level post-execution commands if any.

If both application-level and job-level post-execution commands (`bsub -Ep`) are specified, job level post-execution overrides application-level post-execution commands. Queue-level post-execution commands run after application-level post-execution and job-level post-execution commands

## Chunk job scheduling

`CHUNK_JOB_SIZE` in an application profile ensures that jobs associated with the application are chunked together. `CHUNK_JOB_SIZE=1` disables job chunk scheduling. Application-level job chunk definition overrides chunk job dispatch configured in the queue.

`CHUNK_JOB_SIZE` is ignored and jobs are not chunked under the following conditions:

- ◆ CPU limit greater than 30 minutes (`CPULIMIT` parameter in `lsb.queues` or `lsb.applications`)
- ◆ Run limit greater than 30 minutes (`RUNLIMIT` parameter in `lsb.queues` or `lsb.applications`)
- ◆ Run time estimate greater than 30 minutes (`RUNTIME` parameter in `lsb.applications`)

If `CHUNK_JOB_DURATION` is set in `lsb.params`, chunk jobs are accepted regardless of the value of `CPULIMIT`, `RUNLIMIT` or `RUNTIME`.

## Rerunnable jobs

`RERUNNABLE` in an application profile overrides queue-level job rerun, and allows you to submit rerunnable jobs to a non-rerunnable queue. Job-level rerun (`bsub -r` or `bsub -rn`) overrides both the application profile and the queue.

## Resource requirements

Job-level, application-level, and queue-level resource requirements are merged in the following manner:

- ◆ If resource requirements are not defined at the application level, job-level and queue-level resource requirements are merged.
- ◆ When application-level resource requirements are defined, job-level requirements usually take precedence. Specifically:
  - ❖ The `select` sections from the job, application profile, and queue must all be satisfied.
  - ❖ The job-level `order` section overrides the application profile section, which overrides the queue-level section.
  - ❖ The job-level `usage` section takes precedence. Any `usage` requirements defined in the application profile that are not already specified at the job level are then merged. Any queue-level requirements are then merged with that result.
  - ❖ The job-level `span` section overrides the application profile `span` section, which overrides a queue-level section.
  - ❖ The `same` section from the job, application profile, and queue must all be satisfied.

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements at the job level or application level that exceed the requirements specified in the queue.

If RES\_REQ is defined at the queue level and there are no load thresholds defined, the pending reasons for each individual load index will not be displayed by `bjobs`.

## Estimated runtime and runtime limits

Instead of specifying an explicit runtime limit for jobs, you can specify an *estimated* run time for jobs. LSF uses the estimated value for job scheduling purposes only, and does not kill jobs that exceed this value unless the jobs also exceed a defined runtime limit. The format of runtime estimate is same as run limit set by the `bsub -W` option or the RUNLIMIT parameter in `lsb.queues` and `lsb.applications`.

Use JOB\_RUNLIMIT\_RATIO in `lsb.params` to limit the runtime estimate users can set. If JOB\_RUNLIMIT\_RATIO is set to 0 no restriction is applied to the runtime estimate. The ratio does not apply to the RUNTIME parameter in `lsb.applications`.

The job-level runtime estimate setting overrides the RUNTIME setting in an application profile in `lsb.applications`.

The following LSF features use the estimated runtime value to schedule jobs:

- ◆ Job chunking
- ◆ Advanced reservation
- ◆ SLA
- ◆ Slot reservation
- ◆ Backfill

## Define a runtime estimate

Define the RUNTIME parameter at the application level. Use the `bsub -We` option at the job-level.

You can specify the runtime estimate as hours and minutes, or minutes only. The following examples show an application-level runtime estimate of three hours and 30 minutes:

- ◆ `RUNTIME=3:30`
- ◆ `RUNTIME=210`

## Configuring normalized run time

LSF uses normalized run time for scheduling in order to account for different processing speeds of the execution hosts.

**TIP:** If you want the scheduler to use wall-clock (absolute) run time instead of normalized run time, define `ABS_RUNLIMIT=Y` in the file `lsb.params` or in the file `lsb.applications` for the application associated with your job.

LSF calculates the normalized run time using the following formula:

$$\text{NORMALIZED\_RUN\_TIME} = \text{RUNTIME} * \text{CPU\_Factor\_Normalization\_Host} / \text{CPU\_Factor\_Execute\_Host}$$

You can specify a host name or host model with the runtime estimate so that LSF uses a specific host name or model as the normalization host. If you do not specify a host name or host model, LSF uses the CPU factor for the default normalization host as described in the following table.

| If you define...                                     | In the file... | Then...                                                     |
|------------------------------------------------------|----------------|-------------------------------------------------------------|
| DEFAULT_HOST_SPEC                                    | lsb.queues     | LSF selects the default normalization host for the queue.   |
| DEFAULT_HOST_SPEC                                    | lsb.params     | LSF selects the default normalization host for the cluster. |
| No default host at either the queue or cluster level |                | LSF selects the submission host as the normalization host.  |

To specify a host name (defined in `lsf.cluster.clustername`) or host model (defined in `lsf.shared`) as the normalization host, insert the "/" character between the minutes and the host name or model, as shown in the following examples:

```
RUNTIME=3:30/hostA
bsub -We 3:30/hostA
```

LSF calculates the normalized run time using the CPU factor defined for `hostA`.

```
RUNTIME=210/Ultra5S
bsub -We 210/Ultra5S
```

LSF calculates the normalized run time using the CPU factor defined for host model `Ultra5S`.

**TIP:** Use `lsinfo` to see host name and host model information.

### Guidelines for defining a runtime estimate

- 1 You can define an estimated run time, along with a runtime limit (job level with `bsub -W`, application level with `RUNLIMIT` in `lsb.applications`, or queue level with `RUNLIMIT` in `lsb.queues`).
- 2 If the runtime limit is defined, the job-level (`-We`) or application-level `RUNTIME` value must be less than or equal to the run limit. LSF ignores the estimated runtime value and uses the run limit value for scheduling when
  - ❖ The estimated runtime value exceeds the run limit value, or
  - ❖ An estimated runtime value is not defined

**NOTE:** When LSF uses the run limit value for scheduling, and the run limit is defined at more than one level, LSF uses the smallest run limit value to estimate the job duration.

- 3 For chunk jobs, ensure that the estimated runtime value is
  - ❖ Less than the `CHUNK_JOB_DURATION` defined in the file `lsb.params`, or
  - ❖ Less than 30 minutes, if `CHUNK_JOB_DURATION` is not defined.

## How estimated run time interacts with run limits

The following table includes all the expected behaviors for the combinations of job-level runtime estimate (-We), job-level run limit (-W), application-level runtime estimate (RUNTIME), application-level run limit (RUNLIMIT), queue-level run limit (RUNLIMIT, both default and hard limit). *Ratio* is the value of JOB\_RUNLIMIT\_RATIO defined in `lsb.params`. The dash (—) indicates no value is defined for the job.

| Job-runtime estimate | Job-run limit        | Application runtime estimate | Application run limit | Queue default run limit | Queue hard run limit | Result                                                                                                                                 |
|----------------------|----------------------|------------------------------|-----------------------|-------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| T1                   | -                    | —                            | —                     | —                       | —                    | Job is accepted<br>Jobs running longer than $T1 * ratio$ are killed                                                                    |
| T1                   | $T2 > T1 * ratio$    | —                            | —                     | —                       | —                    | Job is rejected                                                                                                                        |
| T1                   | $T2 \leq T1 * ratio$ | —                            | —                     | —                       | —                    | Job is accepted<br>Jobs running longer than T2 are killed                                                                              |
| T1                   | $T2 \leq T1 * ratio$ | T3                           | T4                    | —                       | —                    | Job is accepted<br>Jobs running longer than T2 are killed<br>T2 overrides T4 or $T1 * ratio$ overrides T4<br>T1 overrides T3           |
| T1                   | $T2 \leq T1 * ratio$ | —                            | —                     | T5                      | T6                   | Job is accepted<br>Jobs running longer than T2 are killed<br>If $T2 > T6$ , the job is rejected                                        |
| T1                   | —                    | T3                           | T4                    | —                       | —                    | Job is accepted<br>Jobs running longer than $T1 * ratio$ are killed<br>T2 overrides T4 or $T1 * ratio$ overrides T4<br>T1 overrides T3 |
| T1                   | —                    | —                            | —                     | T5                      | T6                   | Job is accepted<br>Jobs running longer than $T1 * ratio$ are killed<br>If $T1 * ratio > T6$ , the job is rejected                      |

How application profiles interact with queue and job parameters



## Resource Allocation Limits

### Contents

- ◆ [About Resource Allocation Limits](#) on page 370
- ◆ [Configuring Resource Allocation Limits](#) on page 375
- ◆ [Viewing Information about Resource Allocation Limits](#) on page 384

# About Resource Allocation Limits

## Contents

- ◆ [What resource allocation limits do](#) on page 370
- ◆ [How LSF enforces limits](#) on page 371
- ◆ [How LSF counts resources](#) on page 371
- ◆ [Limits for resource consumers](#) on page 373

## What resource allocation limits do

By default, resource *consumers* like users, hosts, queues, or projects are not limited in the resources available to them for running jobs. *Resource allocation limits* configured in `lsb.resources` restrict:

- ◆ The maximum amount of a resource requested by a job that can be allocated during job scheduling for different classes of jobs to start
- ◆ Which resource consumers the limits apply to

If all of the resource has been consumed, no more jobs can be started until some of the resource is released.

For example, by limiting maximum amount of memory for each of your hosts, you can make sure that your system operates at optimal performance. By defining a memory limit for some users submitting jobs to a particular queue and a specified set of hosts, you can prevent these users from using up all the memory in the system at one time.

### Jobs must specify resource requirements

For limits to apply, the job must specify resource requirements (`bsub -R rusage` string or `RES_REQ` in `lsb.queues`). For example, the a memory allocation limit of 4 MB is configured in `lsb.resources`:

```
Begin Limit
NAME = mem_limit1
MEM = 4
End Limit
```

A job submitted with an `rusage` resource requirement that exceeds this limit:

```
bsub -R "rusage[mem=5]" uname
```

and remains pending:

```
bjobs -p 600
```

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 600   | user1 | PEND | normal | suplin02  |           | uname    | Aug 12 14:05 |

Resource (mem) limit defined cluster-wide has been reached;

A job is submitted with a resource requirement within the configured limit:

```
bsub -R"rusage[mem=3]" sleep 100
```

is allowed to run:

```
bjobs
```

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 600   | user1 | PEND | normal | hostA     |           | uname    | Aug 12 14:05 |

```
604 user1 RUN normal hostA sleep 100 Aug 12 14:09
```

## Resource allocation limits and resource usage limits

Resource allocation limits are not the same as *resource usage limits*, which are enforced during job run time. For example, you set CPU limits, memory limits, and other limits that take effect after a job starts running. See Chapter 33, “[Runtime Resource Usage Limits](#)” for more information.

## How LSF enforces limits

Resource allocation limits are enforced so that they apply to:

- ◆ Several kinds of resources:
  - ❖ Job slots by host
  - ❖ Job slots per processor
  - ❖ Running and suspended jobs
  - ❖ Memory (MB or percentage)
  - ❖ Swap space (MB or percentage)
  - ❖ Tmp space (MB or percentage)
  - ❖ Software licenses
  - ❖ Other shared resources
- ◆ Several kinds of resource consumers:
  - ❖ Users and user groups (all users or per-user)
  - ❖ Hosts and host groups (all hosts or per-host)
  - ❖ Queues (all queues or per-queue)
  - ❖ Projects (all projects or per-project)
- ◆ All jobs in the cluster
- ◆ Combinations of consumers:
  - ❖ For jobs running on different hosts in the same queue
  - ❖ For jobs running from different queues on the same host

## How LSF counts resources

Resources on a host are not available if they are taken by jobs that have been started, but have not yet finished. This means running and suspended jobs count against the limits for queues, users, hosts, projects, and processors that they are associated with.

## Job slot limits

Job slot limits can correspond to the maximum number of jobs that can run at any point in time. For example, a queue cannot start jobs if it has no job slots available, and jobs cannot run on hosts that have no available job slots.

Limits such as such as QJOB\_LIMIT (1sb.queues), HJOB\_LIMIT (1sb.queues), UJOB\_LIMIT (1sb.queues), MXJ (1sb.hosts), JL/U (1sb.hosts), MAX\_JOBS (1sb.users), and MAX\_PEND\_JOBS (1sb.users) limit the number of job slots. When the workload is sequential, job slots are usually equivalent to jobs. For parallel or distributed applications, these are true job slot limits and not job limits.

**Job limits** Job limits, specified by JOBS in a Limit section in `lsb.resources`, correspond to the maximum number of running and suspended jobs that can run at any point in time. If both job limits and job slot limits are configured, the most restrictive limit is applied.

**Resource reservation and backfill** When processor or memory reservation occurs, the reserved resources count against the limits for users, queues, hosts, projects, and processors. When backfilling of parallel jobs occurs, the backfill jobs do not count against any limits.

**MultiCluster** Limits apply only to the cluster where `lsb.resources` is configured. If the cluster leases hosts from another cluster, limits are enforced on those hosts as if they were local hosts.

**Switched jobs can exceed resource allocation limits** If a switched job (`bswitch`) has not been dispatched, then the job behaves as if it were submitted to the new queue in the first place, and the JOBS limit is enforced in the target queue.

If a switched job has been dispatched, then resource allocation limits like SWP, TMP, and JOBS can be exceeded in the target queue. For example, given the following JOBS limit configuration:

```
Begin Limit
USERS QUEUES SLOTS TMP JOBS
- normal - 20 2
- short - 20 2
End Limit
```

Submit 3 jobs to the `normal` queue, and 3 jobs to the `short` queue:

```
bsub -q normal -R"rusage[tmp=20]" sleep 1000
bsub -q short -R"rusage[tmp=20]" sleep 1000
```

`bjobs` shows 1 job in RUN state in each queue:

```
bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
16 user1 RUN normal hosta hosta sleep 1000 Aug 30 16:26
17 user1 PEND normal hosta sleep 1000 Aug 30 16:26
18 user1 PEND normal hosta sleep 1000 Aug 30 16:26
19 user1 RUN short hosta hosta sleep 1000 Aug 30 16:26
20 user1 PEND short hosta sleep 1000 Aug 30 16:26
21 user1 PEND short hosta sleep 1000 Aug 30 16:26
```

`blimits` shows the TMP limit reached:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME USERS QUEUES SLOTS TMP JOBS
NONAME000 - normal - 20/20 1/2
NONAME001 - short - 20/20 1/2
```

Switch the running job in the `normal` queue to the `short` queue:

```
bswitch short 16
```

`bjobs` shows 2 jobs running in the short queue, and the second job running in the normal queue:

`bjobs`

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME   | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|------------|--------------|
| 17    | user1 | RUN  | normal | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 18    | user1 | PEND | normal | hosta     |           | sleep 1000 | Aug 30 16:26 |
| 19    | user1 | RUN  | short  | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 16    | user1 | RUN  | normal | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 20    | user1 | PEND | short  | hosta     |           | sleep 1000 | Aug 30 16:26 |
| 21    | user1 | PEND | short  | hosta     |           | sleep 1000 | Aug 30 16:26 |

`blimits` now shows the TMP limit exceeded and the JOBS limit reached in the short queue:

`blimits`

INTERNAL RESOURCE LIMITS:

| NAME      | USERS | QUEUES | SLOTS | TMP   | JOBS |
|-----------|-------|--------|-------|-------|------|
| NONAME000 | -     | normal | -     | 20/20 | 1/2  |
| NONAME001 | -     | short  | -     | 40/20 | 2/2  |

Switch the running job in the normal queue to the short queue:

`bswitch short 17`

`bjobs` now shows 3 jobs running in the short queue and the third job running in the normal queue:

`bjobs`

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME   | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|------------|--------------|
| 18    | user1 | RUN  | normal | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 19    | user1 | RUN  | short  | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 16    | user1 | RUN  | short  | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 17    | user1 | RUN  | short  | hosta     | hosta     | sleep 1000 | Aug 30 16:26 |
| 20    | user1 | PEND | short  | hosta     |           | sleep 1000 | Aug 30 16:26 |
| 21    | user1 | PEND | short  | hosta     |           | sleep 1000 | Aug 30 16:26 |

`blimits` shows both TMP and JOBS limits exceeded in the short queue:

`blimits`

INTERNAL RESOURCE LIMITS:

| NAME      | USERS | QUEUES | SLOTS | TMP   | JOBS |
|-----------|-------|--------|-------|-------|------|
| NONAME000 | -     | normal | -     | 20/20 | 1/2  |
| NONAME001 | -     | short  | -     | 60/20 | 3/2  |

## Limits for resource consumers

### Host groups

If a limit is specified for a host group, the total amount of a resource used by all hosts in that group is counted. If a host is a member of more than one group, each job running on that host is counted against the limit for all groups to which the host belongs.

### Limits for users and user groups

Jobs are normally queued on a first-come, first-served (FCFS) basis. It is possible for some users to abuse the system by submitting a large number of jobs; jobs from other users must wait until these jobs complete. Limiting resources by user prevents users from monopolizing all the resources.

Users can submit an unlimited number of jobs, but if they have reached their limit for any resource, the rest of their jobs stay pending, until some of their running jobs finish or resources become available.

If a limit is specified for a user group, the total amount of a resource used by all users in that group is counted. If a user is a member of more than one group, each of that user's jobs is counted against the limit for all groups to which that user belongs.

Use the keyword `all` to configure limits that apply to each user or user group in a cluster. This is useful if you have a large cluster but only want to exclude a few users from the limit definition.

### Per-user limits on users and groups

Per-user limits are enforced on each user or individually to each user in the user group listed. If a user group contains a subgroup, the limit also applies to each member in the subgroup recursively.

Per-user limits that use the keywords `all` apply to each user in a cluster. If user groups are configured, the limit applies to each member of the user group, not the group as a whole.

# Configuring Resource Allocation Limits

## Contents

- ◆ [lsb.resources file](#) on page 375
- ◆ [Enable resource allocation limits](#) on page 376
- ◆ [Configure cluster-wide limits](#) on page 376
- ◆ [Compatibility with pre-version 7 job slot limits](#) on page 376
- ◆ [How resource allocation limits map to pre-version 7 job slot limits](#) on page 377
- ◆ [How conflicting limits are resolved](#) on page 378
- ◆ [Example limit configurations](#) on page 380

## lsb.resources file

Configure all resource allocation limits in one or more `Limit` sections in the `lsb.resources` file. `Limit` sections set limits for how much of the specified resources must be available for different classes of jobs to start, and which resource consumers the limits apply to.

## Resource parameters

| To limit ...                                                                                                         | Set in a <code>Limit</code> section of <code>lsb.resources</code> ... |
|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| Total number of running and suspended (RUN, SSUSP, USUSP) jobs                                                       | JOBS                                                                  |
| Total number of job slots that can be used by specific jobs                                                          | SLOTS                                                                 |
| Jobs slots based on the number of processors on each host affected by the limit                                      | SLOTS_PER_PROCESSOR and PER_HOST                                      |
| Memory— if PER_HOST is set for the limit, the amount can be a percentage of memory on each host in the limit         | MEM (MB or percentage)                                                |
| Swap space —if PER_HOST is set for the limit, the amount can be a percentage of swap space on each host in the limit | SWP (MB or percentage)                                                |
| Tmp space — if PER_HOST is set for the limit, the amount can be a percentage of tmp space on each host in the limit  | TMP (MB or percentage)                                                |
| Software licenses                                                                                                    | LICENSE or RESOURCE                                                   |
| Any shared resource                                                                                                  | RESOURCE                                                              |

## Consumer parameters

|                                                                    |                                                          |
|--------------------------------------------------------------------|----------------------------------------------------------|
| For jobs submitted ...                                             | Set in a Limit section of <code>lsb.resources</code> ... |
| By all specified users or user groups                              | USERS                                                    |
| To all specified queues                                            | QUEUES                                                   |
| To all specified hosts or host groups                              | HOSTS                                                    |
| For all specified projects                                         | PROJECTS                                                 |
| By each specified user or each member of the specified user groups | PER_USER                                                 |
| To each specified queue                                            | PER_QUEUE                                                |
| To each specified host or each member of the specified host groups | PER_HOST                                                 |
| For each specified project                                         | PER_PROJECT                                              |

## Enable resource allocation limits

- To enable resource allocation limits in your cluster, you configure the resource allocation limits scheduling plugin `schmod_limit` in `lsb.modules`:
 

```
Begin PluginModule
SCH_PLUGIN RB_PLUGIN SCH_DISABLE_PHASES
schmod_default () ()
schmod_limit () ()
End PluginModule
```

## Configure cluster-wide limits

- To configure limits that take effect for your entire cluster, configure limits in `lsb.resources`, but do not specify any consumers.

## Compatibility with pre-version 7 job slot limits

The `Limit` section of `lsb.resources` does not support the keywords or format used in `lsb.users`, `lsb.hosts`, and `lsb.queues`. However, any existing job slot limit configuration in these files will continue to apply.



## How resource allocation limits map to pre-version 7 job slot limits

Job slot limits are the only type of limit you can configure in `lsb.users`, `lsb.hosts`, and `lsb.queues`. You cannot configure limits for user groups, host groups, and projects in `lsb.users`, `lsb.hosts`, and `lsb.queues`. You should not configure any new resource allocation limits in `lsb.users`, `lsb.hosts`, and `lsb.queues`. Use `lsb.resources` to configure all new resource allocation limits, including job slot limits.

| Job slot resources<br>(lsb.resources) | Resource consumers (lsb.resources) |          |                   |                  |          | Equivalent existing limit<br>(file) |
|---------------------------------------|------------------------------------|----------|-------------------|------------------|----------|-------------------------------------|
|                                       | USERS                              | PER_USER | QUEUES            | HOSTS            | PER_HOST |                                     |
| SLOTS                                 | —                                  | all      | —                 | <i>host_name</i> | —        | JL/U<br>(lsb.hosts)                 |
| SLOTS_PER_PROCESSOR                   | <i>user_name</i>                   | —        | —                 | —                | all      | JL/P (lsb.users)                    |
| SLOTS                                 | —                                  | all      | <i>queue_name</i> | —                | —        | UJOB_LIMIT<br>(lsb.queues)          |
| SLOTS                                 | —                                  | all      | —                 | —                | —        | MAX_JOBS<br>(lsb.users)             |
| SLOTS                                 | —                                  | —        | <i>queue_name</i> | —                | all      | HJOB_LIMIT<br>(lsb.queues)          |
| SLOTS                                 | —                                  | —        | —                 | <i>host_name</i> | —        | MXJ (lsb.hosts)                     |
| SLOTS_PER_PROCESSOR                   | —                                  | —        | <i>queue_name</i> | —                | all      | PJOB_LIMIT<br>(lsb.queues)          |
| SLOTS                                 | —                                  | —        | <i>queue_name</i> | —                | —        | QJOB_LIMIT<br>(lsb.queues)          |

Limits for the following resources have no corresponding limit in `lsb.users`, `lsb.hosts`, and `lsb.queues`:

- ◆ JOBS
- ◆ LICENSE
- ◆ RESOURCE
- ◆ SWP
- ◆ TMP

## How conflicting limits are resolved

LSF handles two kinds of limit conflicts:

- ◆ [Similar conflicting limits](#)
- ◆ [Equivalent conflicting limits](#)

### Similar conflicting limits

For similar limits configured in `lsb.resources`, `lsb.users`, `lsb.hosts`, or `lsb.queues`, the most restrictive limit is used. For example, a slot limit of 3 for all users is configured in `lsb.resources`:

```
Begin Limit
NAME = user_limit1
USERS = all
SLOTS = 3
End Limit
```

This is similar, but *not equivalent* to an existing `MAX_JOBS` limit of 2 is configured in `lsb.users`.

**busers**

| USER/GROUP | JL/P | MAX | NJOBS | PEND | RUN | SSUSP | USUSP | RSV |
|------------|------|-----|-------|------|-----|-------|-------|-----|
| user1      | -    | 2   | 4     | 2    | 2   | 0     | 0     | 0   |

user1 submits 4 jobs:

**bjobs**

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME   | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|------------|--------------|
| 816   | user1 | RUN  | normal | hostA     | hostA     | sleep 1000 | Jan 22 16:34 |
| 817   | user1 | RUN  | normal | hostA     | hostA     | sleep 1000 | Jan 22 16:34 |
| 818   | user1 | PEND | normal | hostA     |           | sleep 1000 | Jan 22 16:34 |
| 819   | user1 | PEND | normal | hostA     |           | sleep 1000 | Jan 22 16:34 |

Two jobs (818 and 819) remain pending because the more restrictive limit of 2 from `lsb.users` is enforced:

**bjobs -p**

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | JOB_NAME   | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|------------|--------------|
| 818   | user1 | PEND | normal | hostA     | sleep 1000 | Jan 22 16:34 |

The user has reached his/her job slot limit;

|     |       |      |        |       |            |              |
|-----|-------|------|--------|-------|------------|--------------|
| 819 | user1 | PEND | normal | hostA | sleep 1000 | Jan 22 16:34 |
|-----|-------|------|--------|-------|------------|--------------|

The user has reached his/her job slot limit;

If the `MAX_JOBS` limit in `lsb.users` is 4:

**busers**

| USER/GROUP | JL/P | MAX | NJOBS | PEND | RUN | SSUSP | USUSP | RSV |
|------------|------|-----|-------|------|-----|-------|-------|-----|
| user1      | -    | 4   | 4     | 1    | 3   | 0     | 0     | 0   |

and user1 submits 4 jobs:

**bjobs**

| JOBID | USER  | STAT | QUEUE  | FROM_HOST | EXEC_HOST | JOB_NAME   | SUBMIT_TIME  |
|-------|-------|------|--------|-----------|-----------|------------|--------------|
| 824   | user1 | RUN  | normal | hostA     | hostA     | sleep 1000 | Jan 22 16:38 |

```

825 user1 RUN normal hostA hostA sleep 1000 Jan 22 16:38
826 user1 RUN normal hostA hostA sleep 1000 Jan 22 16:38
827 user1 PEND normal hostA sleep 1000 Jan 22 16:38

```

Only one job (827) remains pending because the more restrictive limit of 3 in `lsb.resources` is enforced:

```
bjobs -p
```

```

JOBID USER STAT QUEUE FROM_HOST JOB_NAME SUBMIT_TIME
827 user1 PEND normal hostA sleep 1000 Jan 22 16:38

```

Resource (slot) limit defined cluster-wide has been reached;

## Equivalent conflicting limits

New limits in `lsb.resources` that are equivalent to existing limits in `lsb.users`, `lsb.hosts`, or `lsb.queues`, but with a different value override the existing limits. The equivalent limits in `lsb.users`, `lsb.hosts`, or `lsb.queues` are ignored, and the value of the new limit in `lsb.resources` is used.

For example, a *per-user* job slot limit in `lsb.resources` is equivalent to a `MAX_JOBS` limit in `lsb.users`, so only the `lsb.resources` limit is enforced, the limit in `lsb.users` is ignored:

```

Begin Limit
NAME = slot_limit
PER_USER =all
SLOTS = 3
End Limit

```

## How job limits work

The `JOBS` parameter limits the maximum number of running or suspended jobs available to resource consumers. Limits are enforced depending on the number of jobs in `RUN`, `SSUSP`, and `USUSP` state.

## Stopping and resuming jobs

Jobs stopped with `bstop`, go into `USUSP` status. LSF includes `USUSP` jobs in the count of running jobs, so the usage of `JOBS` limit will not change when you suspend a job.

Resuming a stopped job (`brresume`) changes job status to `SSUSP`. The job can enter `RUN` state, if the `JOBS` limit has not been exceeded. Lowering the `JOBS` limit before resuming the job can exceed the `JOBS` limit, and prevent `SSUSP` jobs from entering `RUN` state.

For example, `JOBS=5`, and 5 jobs are running in the cluster (`JOBS` has reached 5/5). Normally, the stopped job (in `USUSP` state) can later be resumed and begin running, returning to `RUN` state. If you reconfigure the `JOBS` limit to 4 before resuming the job, the `JOBS` usage becomes 5/4, and the job cannot run because the `JOBS` limit has been exceeded.

## Preemption

The `JOBS` limit does not block preemption based on job slots. For example, if `JOBS=2`, and a host is already running 2 jobs in a preemptable queue, a new preemptive job can preempt a job on that host as long as the preemptive slots can be satisfied even though the `JOBS` limit has been reached.

### Reservation and backfill

Reservation and backfill are still made at the job slot level, but despite a slot reservation being satisfied, the job may ultimately not run because the JOBS limit has been reached. This is similar to a job not running because a license is not available.

### Other jobs

- ◆ `brun` forces a pending job to run immediately on specified hosts. A job forced to run with `brun` is counted as a running job, which may violate JOBS limits. After the forced job starts, the JOBS limits may be exceeded.
- ◆ Requeued jobs (`brequeue`) are assigned PENDING or PSUSP. Usage of JOBS limit is decreased by the number of requeued jobs.
- ◆ Checkpointed jobs restarted with `brstart` start a new job based on the checkpoint of an existing job. Whether the new job can run depends on the limit policy (including the JOBS limit) that applies to the job. For example, if you checkpoint a job running on a host that has reached its JOBS limit, then restart it, the restarted job cannot run because the JOBS limit has been reached.
- ◆ For job arrays, you can define a maximum number of jobs that can run in the array at any given time. The JOBS limit, like other resource allocation limits, works in combination with the array limits. For example, if JOBS=3 and the array limit is 4, at most 3 job elements can run in the array.
- ◆ For chunk jobs, only the running job among the jobs that are dispatched together in a chunk is counted against the JOBS limit. Jobs in WAIT state do not affect the JOBS limit usage.

## Example limit configurations

Each set of limits is defined in a `Limit` section enclosed by `Begin Limit` and `End Limit`.

### Example 1

`user1` is limited to 2 job slots on `hostA`, and `user2`'s jobs on queue `normal` are limited to 20 MB of memory:

```
Begin Limit
NAME HOSTS SLOTS MEM SWP TMP USERS QUEUES
Limit1 hostA 2 - - - user1 -
- - - 20 - - user2 normal
End Limit
```

### Example 2

Set a job slot limit of 2 for user `user1` submitting jobs to queue `normal` on host `hostA` for all projects, but only one job slot for all queues and hosts for project `test`:

```
Begin Limit
HOSTS SLOTS PROJECTS USERS QUEUES
hostA 2 - user1 normal
- 1 test user1 -
End Limit
```

### Example 3

Limit usage of hosts in `license1` group:

- ◆ 10 jobs can run from `normal` queue
- ◆ Any number can run from `short` queue, but only can use 200 MB of memory in total

- ◆ Each other queue can run 30 jobs, each queue using up to 300 MB of memory in total

```

Begin Limit
HOSTS SLOTS MEM PER_QUEUE
license1 10 - normal
license1 - 200 short
license1 30 300 (all ~normal ~short)
End Limit

```

#### Example 4

All users in user group `ugroup1` except `user1` using `queue1` and `queue2` and running jobs on hosts in host group `hgroup1` are limited to 2 job slots per processor on each host:

```

Begin Limit
NAME = limit1
Resources:
SLOTS_PER_PROCESSOR = 2
#Consumers:
QUEUES = queue1 queue2
USERS = ugroup1 ~user1
PER_HOST = hgroup1
End Limit

```

#### Example 5

`user1` and `user2` can use all queues and all hosts in the cluster with a limit of 20 MB of available memory:

```

Begin Limit
NAME = 20_MB_mem
Resources:
MEM = 20
Consumers:
USERS = user1 user2
End Limit

```

#### Example 6

All users in user group `ugroup1` can use `queue1` and `queue2` and run jobs on any host in host group `hgroup1` sharing 10 job slots:

```

Begin Limit
NAME = 10_slot
Resources:
SLOTS = 10
#Consumers:
QUEUES = queue1 queue2
USERS = ugroup1
HOSTS = hgroup1
End Limit

```

## Configuring Resource Allocation Limits

### Example 7

All users in user group `ugroup1` except `user1` can use all queues but `queue1` and run jobs with a limit of 10% of available memory on each host in host group

```
hgroup1:
Begin Limit
NAME = 10_percent_mem
Resources:
MEM = 10%
QUEUES = all ~queue1
USERS = ugroup1 ~user1
PER_HOST = hgroup1
End Limit
```

### Example 8

Limit users in the `develop` group to 1 job on each host, and 50% of the memory on the host.

```
Begin Limit
NAME = develop_group_limit
Resources:
SLOTS = 1
MEM = 50%
#Consumers:
USERS = develop
PER_HOST = all
End Limit
```

### Example 9

Limit software license `lic1`, with quantity 100, where `user1` can use 90 licenses and all other users are restricted to 10.

```
Begin Limit
USERS LICENSE
user1 ([lic1,90])
(all ~user1) ([lic1,10])
End Limit
```

`lic1` is defined as a decreasing numeric shared resource in `lsf.shared`.

To submit a job to use one `lic1` license, use the `rusage` string in the `-R` option of `bsub` specify the license:

```
bsub -R "rusage[lic1=1]" my-job
```

### Example 10

Jobs from `crash` project can use 10 `lic1` licenses, while jobs from all other projects together can use 5.

```
Begin Limit
LICENSE PROJECTS
([lic1,10]) crash
([lic1,5]) (all ~crash)
End Limit
```

`lic1` is defined as a decreasing numeric shared resource in `lsf.shared`.

**Example 11**

Limit all hosts to 1 job slot per processor:

```
Begin Limit
NAME = default_limit
SLOTS_PER_PROCESSOR = 1
PER_HOST = all
End Limit
```

**Example 12**

The short queue can have at most 200 running and suspended jobs:

```
Begin Limit
NAME = shortq_limit
QUEUES = short
JOBS = 200
End Limit
```

## Viewing Information about Resource Allocation Limits

Your job may be pending because some configured resource allocation limit has been reached. Use the `blimits` command to show the dynamic counters of resource allocation limits configured in Limit sections in `lsb.resources`. `blimits` displays the current resource usage to show what limits may be blocking your job.

### blimits command

The `blimits` command displays:

- ◆ Configured limit policy name
- ◆ Users (`-u` option)
- ◆ Queues (`-q` option)
- ◆ Hosts (`-m` option)
- ◆ Project names (`-P` option)
- ◆ Limits (SLOTS, MEM, TMP, SWP, JOBS)
- ◆ All resource configurations in `lsb.resources` (`-c` option). This is the same as `bresources` with no options.

Resources that have no configured limits or no limit usage are indicated by a dash (-). Limits are displayed in a USED/LIMIT format. For example, if a limit of 10 slots is configured and 3 slots are in use, then `blimits` displays the limit for SLOTS as 3/10.

If limits MEM, SWP, or TMP are configured as percentages, both the limit and the amount used are displayed in MB. For example, `lshosts` displays `maxmem` of 249 MB, and MEM is limited to 10% of available memory. If 10 MB out of 25 MB are used, `blimits` displays the limit for MEM as 10/25 (10 MB USED from a 25 MB LIMIT).

Configured limits and resource usage for built-in resources (slots, mem, tmp, and swp load indices, and number of running and suspended jobs) are displayed as INTERNAL RESOURCE LIMITS separately from custom external resources, which are shown as EXTERNAL RESOURCE LIMITS.

Limits are displayed for both the vertical tabular format and the horizontal format for Limit sections. If a vertical format Limit section has no name, `blimits` displays `NONAME $nnn$`  under the NAME column for these limits, where the unnamed limits are numbered in the order the vertical-format Limit sections appear in the `lsb.resources` file.

If a resource consumer is configured as `all`, the limit usage for that consumer is indicated by a dash (-).

PER\_HOST slot limits are not displayed. The `bhosts` commands displays these as MXJ limits.

In MultiCluster, `blimits` returns the information about all limits in the local cluster.



## Examples

For the following limit definitions:

```

Begin Limit
NAME = limit1
USERS = user1
PER_QUEUE = all
PER_HOST = hostA hostC
TMP = 30%
SWP = 50%
MEM = 10%
End Limit
Begin Limit
NAME = limit_ext1
PER_HOST = all
RESOURCE = ([user1_num,30] [hc_num,20])
End Limit
Begin Limit
NAME = limit2
QUEUES = short
JOBS = 200
End Limit

```

`blimits` displays the following:

`blimits`

INTERNAL RESOURCE LIMITS:

| NAME   | USERS | QUEUES | HOSTS          | PROJECTS | SLOTS | MEM   | TMP     | SWP    | JOBS   |
|--------|-------|--------|----------------|----------|-------|-------|---------|--------|--------|
| limit1 | user1 | q2     | hostA@cluster1 | -        | -     | 10/25 | -       | 10/258 | -      |
| limit1 | user1 | q3     | hostA@cluster1 | -        | -     | -     | 30/2953 | -      | -      |
| limit1 | user1 | q4     | hostC          | -        | -     | -     | 40/590  | -      | -      |
| limit2 | -     | short  | -              | -        | -     | -     | -       | -      | 50/200 |

EXTERNAL RESOURCE LIMITS:

| NAME       | USERS | QUEUES | HOSTS          | PROJECTS | user1_num | hc_num |
|------------|-------|--------|----------------|----------|-----------|--------|
| limit_ext1 | -     | -      | hostA@cluster1 | -        | -         | 1/20   |
| limit_ext1 | -     | -      | hostC@cluster1 | -        | 1/30      | 1/20   |

- ◆ In limit policy `limit1`, `user1` submitting jobs to `q2`, `q3`, or `q4` on `hostA` or `hostC` is limited to 30% tmp space, 50% swap space, and 10% available memory. No limits have been reached, so the jobs from `user1` should run. For example, on `hostA` for jobs from `q2`, 10 MB of memory are used from a 25 MB limit and 10 MB of swap space are used from a 258 MB limit.

## Viewing Information about Resource Allocation Limits

- ◆ In limit policy `limit_ext1`, external resource `user1_num` is limited to 30 per host and external resource `hc_num` is limited to 20 per host. Again, no limits have been reached, so the jobs requesting those resources should run.
- ◆ In limit policy `limit2`, the `short` queue can have at most 200 running and suspended jobs. 50 jobs are running or suspended against the 200 job limit. The limit has not been reached, so jobs can run in the `short` queue.

## Reserving Resources

### Contents

- ◆ [About Resource Reservation](#) on page 387
- ◆ [Using Resource Reservation](#) on page 388
- ◆ [Memory Reservation for Pending Jobs](#) on page 390
- ◆ [Time-based Slot Reservation](#) on page 392
- ◆ [Viewing Resource Reservation Information](#) on page 399

### About Resource Reservation

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs do not consume the resources they require when they first start. Instead, they will typically use the resources over a period of time.

For example, a job requiring 100 MB of swap is dispatched to a host having 150 MB of available swap. The job starts off initially allocating 5 MB and gradually increases the amount consumed to 100 MB over a period of 30 minutes. During this period, another job requiring more than 50 MB of swap should not be started on the same host to avoid over-committing the resource.

Resources can be reserved to prevent overcommitment by LSF. Resource reservation requirements can be specified as part of the resource requirements when submitting a job, or can be configured into the queue level resource requirements.

### How resource reservation works

When deciding whether to schedule a job on a host, LSF considers the reserved resources of jobs that have previously started on that host. For each load index, the amount reserved by all jobs on that host is summed up and subtracted (or added if the index is increasing) from the current value of the resources as reported by the LIM to get amount available for scheduling new jobs:

available amount = current value - reserved amount for all jobs

## Using Resource Reservation

For example:

```
bsub -R "rusage[tmp=30:duration=30:decay=1]" myjob
```

will reserve 30 MB of temp space for the job. As the job runs, the amount reserved will decrease at approximately 1 MB/minute such that the reserved amount is 0 after 30 minutes.

## Queue-level and job-level resource reservation

The queue level resource requirement parameter `RES_REQ` may also specify the resource reservation. If a queue reserves certain amount of a resource, you cannot reserve a greater amount of that resource at the job level.

For example, if the output of `bqueues -l` command contains:

```
RES_REQ: rusage[mem=40:swp=80:tmp=100]
```

the following submission will be rejected since the requested amount of certain resources exceeds queue's specification:

```
bsub -R "rusage[mem=50:swp=100]" myjob
```

## Using Resource Reservation

### Queue-level resource reservation

At the queue level, resource reservation allows you to specify the amount of resources to reserve for jobs in the queue. It also serves as the upper limits of resource reservation if a user also specifies it when submitting a job.

### Queue-level resource reservation and pending reasons

The use of `RES_REQ` affects the pending reasons as displayed by `bjobs`. If `RES_REQ` is specified in the queue and the `loadSched` thresholds are not specified, then the pending reasons for each individual load index will not be displayed.

### Configuring resource reservation at the queue level

Queue-level resource reservation can be configured as part of the `RES_REQ` parameter. The resource reservation requirement can be configured at the queue level as part of the queue level resource requirements. Use the resource usage (`rusage`) section of the resource requirement string to specify the amount of resources a job should reserve after it is started.

### Examples

```
Begin Queue
.
RES_REQ = select[type==any] rusage[swp=100:mem=40:duration=60]
.
End Queue
```

This will allow a job to be scheduled on any host that the queue is configured to use and will reserve 100 MB of swap and 40 MB of memory for a duration of 60 minutes.

```
Begin Queue
.
RES_REQ = select[type==any] rusage[mem=20||mem=10:swp=20]
.
End Queue
```

This will allow a job to be scheduled on any host that the queue is configured to use. The job will attempt to reserve 20 MB of memory, or 10 MB of memory and 20 MB of swap if the 20 MB of memory is unavailable.

## Job-level resource reservation

- 1 To specify resource reservation at the job level, use `bsub -R` and include the resource usage section in the resource requirement string.

## Configure per-resource reservation

- 1 To enable greater flexibility for reserving numeric resources are reserved by jobs, configure the `ReservationUsage` section in `lsb.resources` to reserve resources like license tokens per resource as `PER_JOB`, `PER_SLOT`, or `PER_HOST`:

```
Begin ReservationUsage
RESOURCE METHOD
licenseX PER_JOB
licenseY PER_HOST
licenseZ PER_SLOT
End ReservationUsage
```

Only user-defined numeric resources can be reserved. Builtin resources like `mem`, `cpu`, `swp`, etc. cannot be configured in the `ReservationUsage` section.

The cluster-wide `RESOURCE_RESERVE_PER_SLOT` parameter in `lsb.params` is obsolete. Configuration in `lsb.resources` overrides `RESOURCE_RESERVE_PER_SLOT` if it also exists for the same resource.

`RESOURCE_RESERVE_PER_SLOT` parameter still controls resources not configured in `lsb.resources`. Resources not reserved in `lsb.resources` are reserved per job.

`PER_HOST` reservation means that for the parallel job, LSF reserves one instance of a for each host. For example, some application licenses are charged only once no matter how many applications are running provided those applications are running on the same host under the same user.

## Assumptions and limitations

- ◆ Per-resource configuration defines resource usage for individual resources, but it does not change any existing resource limit behavior (`PER_JOB`, `PER_SLOT`).
- ◆ In a MultiCluster environment, you should configure resource usage in the scheduling cluster (submission cluster in lease model or receiving cluster in job forward model).

## Memory Reservation for Pending Jobs

### About memory reservation for pending jobs

By default, the `usage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available, until the total required memory specified on the `usage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

### Configure memory reservation for pending jobs

#### RESOURCE\_RESERVE parameter

- 1 Use the `RESOURCE_RESERVE` parameter in `lsb.queues` to reserve host memory for pending jobs.

The amount of memory reserved is based on the currently available memory when the job is pending. Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of `MAX_RESERVE_TIME` set on the `RESOURCE_RESERVE` parameter.

#### Configure lsb.modules

- 1 To enable memory reservation for sequential jobs, add the LSF scheduler plugin module name for resource reservation (`schmod_reserve`) to the `lsb.modules` file:

```
Begin PluginModule
SCH_PLUGIN RB_PLUGIN SCH_DISABLE_PHASES
schmod_default () ()
schmod_reserve () ()
schmod_preemption () ()
End PluginModule
```

#### Configure lsb.queues

- 1 Set the `RESOURCE_RESERVE` parameter in a queue defined in `lsb.queues`. If both `RESOURCE_RESERVE` and `SLOT_RESERVE` are defined in the same queue, job slot reservation and memory reservation are both enabled and an error is displayed when the cluster is reconfigured. `SLOT_RESERVE` is ignored.

## Example queues

The following queue enables memory reservation for pending jobs:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

## Use memory reservation for pending jobs

- 1 Use the `rusage` string in the `-R` option to `bsub` or the `RES_REQ` parameter in `lsb.queues` to specify the amount of memory required for the job. Submit the job to a queue with `RESOURCE_RESERVE` configured.

See [Examples](#) on page 398 for examples of jobs that use memory reservation.

## How memory reservation for pending jobs works

### Amount of memory reserved

The amount of memory reserved is based on the currently available memory when the job is pending. For example, if LIM reports that a host has 300 MB of memory available, the job submitted by the following command:

```
bsub -R "rusage[mem=400]" -q reservation my_job
```

will be pending and reserve the 300 MB of available memory. As other jobs finish, the memory that becomes available is added to the reserved memory until 400 MB accumulates, and the job starts.

No memory is reserved if no job slots are available for the job because the job could not run anyway, so reserving memory would waste the resource.

Only memory is accumulated while the job is pending; other resources specified on the `rusage` string are only reserved when the job is running. Duration and decay have no effect on memory reservation while the job is pending.

### How long memory is reserved (MAX\_RESERVE\_TIME)

Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of `MAX_RESERVE_TIME` set on the `RESOURCE_RESERVE` parameter. If a job has not accumulated enough memory to start by the time `MAX_RESERVE_TIME` expires, it releases all its reserved memory so that other pending jobs can run. After the reservation time expires, the job cannot reserve slots or memory for one scheduling session, so other jobs have a chance to be dispatched. After one scheduling session, the job can reserve available resources again for another period specified by `MAX_RESERVE_TIME`.

### Examples

#### lsb.queues

The following queues are defined in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

#### Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

#### Sequential jobs

Each of the following sequential jobs requires 400 MB of memory and runs for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements yields the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserves all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

## Time-based Slot Reservation

Existing LSF slot reservation works in simple environments, where host-based MXJ limit is only constraint to job slot request. In complex environments, where more than one constraints exist, for example job topology or generic slot limit:

- ◆ Estimated job start time becomes inaccurate
- ◆ The scheduler makes a reservation decision that can postpone estimated job start time or decrease cluster utilization.

Current slot reservation by start time (`RESERVE_BY_STARTTIME`) resolves several reservation issues in multiple candidate host groups, but it cannot help on other cases:

- ◆ Special topology requests, like `span[ptile=n]`
- ◆ Only calculates and displays reservation if host has free slots. Reservations may change or disappear if there are no free CPUs; for example, if a backfill job takes all reserved CPUs.
- ◆ For HPC machines containing many internal nodes, host-level number of reserved slots is not enough for administrator and end user to tell which CPUs the job is reserving and waiting for.



## Time-based slot reservation vs. greedy slot reservation

With time-based reservation, a set of pending jobs will get future allocation and an estimated start time, so that job can be placed in the future. Reservations are made based on future allocation to guarantee estimated start time.

### Start time and future allocation

The estimated start time for a future allocation is the earliest start time when all considered job constraints are satisfied in the future. There may be a small delay of a few minutes between the job finish time on which the estimate was based and the actual start time of the allocated job.

If a job cannot be placed in a future allocation, the scheduler uses *greedy* slot reservation to reserve slots. Existing LSF slot reservation is a simple greedy algorithm:

- ◆ Only considers current available resources and minimal number of requested job slots to reserve as many slots as it is allowed
- ◆ For multiple exclusive candidate host groups, scheduler goes through those groups and makes reservation on the group that has the largest available slots
- ◆ For estimated start time, after making reservation, scheduler sorts all running jobs in ascending order based on their finish time and goes through this sorted job list to add up slots used by each running job till it satisfies minimal job slots request. The finish time of last visited job will be job estimated start time.

Reservation decisions made by greedy slot reservation will not have an accurate estimated start time or information about future allocation. The calculated job start time used for backfill scheduling is uncertain, so `bjobs` displays:

```
Job will start no sooner than indicated time stamp
```

### Time-based reservation and greedy reservation compared

| Start time prediction                                          | Time-based reservation                          | Greedy reservation |
|----------------------------------------------------------------|-------------------------------------------------|--------------------|
| Backfill scheduling if free slots are available                | Yes                                             | Yes                |
| Correct with no job topology                                   | Yes                                             | Yes                |
| Correct for job topology requests                              | Yes                                             | No                 |
| Correct based on resource allocation limits                    | Yes (guaranteed if only two limits are defined) | No                 |
| Correct for memory requests                                    | Yes                                             | No                 |
| When no slots are free for reservation                         | Yes                                             | No                 |
| Future allocation and reservation based on earliest start time | Yes                                             | No                 |
| <code>bjobs</code> displays best estimate                      | Yes                                             | No                 |
| <code>bjobs</code> displays predicted future allocation        | Yes                                             | No                 |

| Start time prediction                      | Time-based reservation | Greedy reservation |
|--------------------------------------------|------------------------|--------------------|
| Absolute predicted start time for all jobs | No                     | No                 |
| Advance reservation considered             | No                     | No                 |

### Greedy reservation example

A cluster has four hosts: A, B, C and D, with 4 CPUs each. Four jobs are running in the cluster: Job1, Job2, Job3 and Job4. According to calculated job estimated start time, the job finish times (FT) have this order:  $FT(\text{Job2}) < FT(\text{Job4}) < FT(\text{Job1}) < FT(\text{Job3})$ .



The highest priority pending job requests `-n 6 -R "span[ptile=2]"`. Based on this, the job needs three hosts and two CPUs on each host. The default greedy slot reservation calculates job start time as the job finish time of Job4; after Job4 finishes, requested slots of pending job can be satisfied.

### Configuring time-based slot reservation

Greedy slot reservation is the default slot reservation mechanism and time-based slot reservation is disabled.


#### LSB\_TIME\_RESERVE\_NUMJOBS (lsf.conf)

- Use `LSB_TIME_RESERVE_NUMJOBS=maximum_reservation_jobs` in `lsf.conf` to enable time-based slot reservation. The value must be a positive integer.  
 LSB\_TIME\_RESERVE\_NUMJOBS controls maximum number of jobs using time-based slot reservation. For example, if `LSB_TIME_RESERVE_NUMJOBS=4`, only the top 4 jobs will get their future allocation information.
- Use `LSB_TIME_RESERVE_NUMJOBS=1` to allow only the highest priority job to get accurate start time prediction.  
 Smaller values are better than larger values because after the first pending job starts, the estimated start time of remaining jobs may be changed. For example, you could configure `LSB_TIME_RESERVE_NUMJOBS` based on the number of exclusive host partitions or host groups.

## Some scheduling examples

- 1 Job5 requests `-n 6 -R "span[ptile=2]"`, which will require three hosts with 2 CPUs on each host. As in the greedy slot reservation example, four jobs are running in the cluster: Job1, Job2, Job3 and Job4. Two CPUs are available now, 1 on host A, and 1 on host D:

| A    | B    | C    | D    |
|------|------|------|------|
| Job4 | Job3 | Job4 | Job3 |
| Job4 | Job3 | Job4 | Job3 |
| Job2 | Job3 | Job2 |      |
|      | Job3 | Job1 | Job1 |



Current Inuse

Current available

- 2 Job2 finishes, freeing 2 more CPUs for future allocation, 1 on host A, and 1 on host C:

| A    | B    | C    | D    |
|------|------|------|------|
| Job4 | Job3 | Job4 | Job3 |
| Job4 | Job3 | Job4 | Job3 |
|      | Job3 |      |      |
|      | Job3 | Job1 | Job1 |



Future available

Current Inuse

Current available

- 3 Job4 finishes, freeing 4 more CPUs for future allocation, 2 on host A, and 2 on host C:

| A | B    | C    | D    |
|---|------|------|------|
|   | Job3 |      | Job3 |
|   | Job3 |      | Job3 |
|   | Job3 |      |      |
|   | Job3 | Job1 | Job1 |



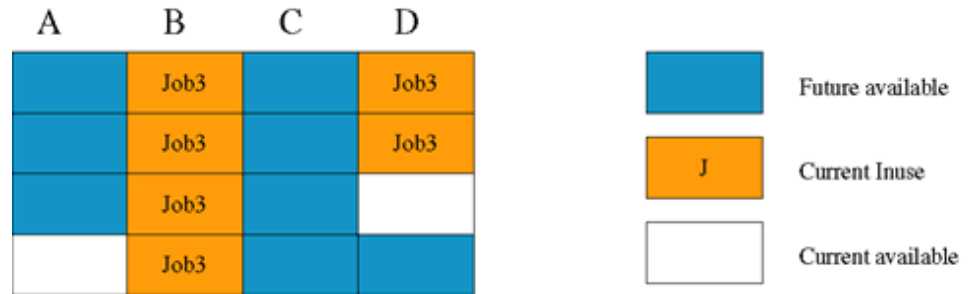
Future available

Current Inuse

Current available

- 4 Job1 finishes, freeing 2 more CPUs for future allocation, 1 on host C, and 1 on host D:

## Time-based Slot Reservation



5 Job5 can now be placed with 2 CPUs on host A, 2 CPUs on host C, and 2 CPUs on host D. The estimated start time is shown as the finish time of Job1:



## Assumptions and limitations

- ◆ To get an accurate estimated start time, you must specify a run limit at the job level using the `bsub -W` option, in the queue by configuring `RUNLIMIT` in `lsb.queues`, or in the application by configuring `RUNLIMIT` in `lsb.applications`, or you must specify a run time estimate by defining the `RUNTIME` parameter in `lsb.applications`. If a run limit or a run time estimate is not defined, the scheduler will try to use CPU limit instead.
- ◆ Estimated start time is only relatively accurate according to current running job information. If running jobs finish earlier, estimated start time may be moved to earlier time. Only the highest priority job will get accurate predicted start time. The estimated start time for other jobs could be changed after the first job starts.
- ◆ Under time-based slot reservation, only information from currently running jobs is used for making reservation decisions.
- ◆ Estimated start time calculation does not consider Deadline scheduling.
- ◆ Estimated start time calculation does not consider Advance Reservation.
- ◆ Estimated start time calculation does not consider `DISPATCH_WINDOW` in `lsb.hosts` and `lsb.queue` configuration.
- ◆ If preemptive scheduling is used, the estimated start time may not be accurate. The scheduler may calculate and estimated time, but actually it may preempt other jobs to start earlier.

## Slot limit enforcement

The following slot limits are enforced:

- ◆ Slot limits configured in `lsb.resources` (`SLOTS`, `PER_SLOT`)

- ◆ MXJ, JL/U in `lsb.hosts`
- ◆ PJOB\_LIMIT, HJOB\_LIMIT, QJOB\_LIMIT, UJOB\_LIMIT in `lsb.queues`

## Memory request

To request memory resources, configure `RESOURCE_RESERVE` in `lsb.queues`. When `RESOURCE_RESERVE` is used, LSF will consider memory and slot requests during time-based reservation calculation. LSF will not reserve slot or memory if any other resources are not satisfied.

If `SLOT_RESERVE` is configured, time-based reservation will not make a slot reservation if any other type of resource is not satisfied, including memory requests.

When `SLOT_RESERVE` is used, if job cannot run because of non-slot resources, including memory, time-based reservation will not reserve slots. For example, if job cannot run because it cannot get required license, job will be pending without any reservation

## Host partition and queue-level scheduling

If host partitions are configured, LSF first schedules jobs on the host partitions and then goes through each queue to schedule jobs. The same job may be scheduled several times, one for each host partition and last one at queue-level. Available candidate hosts may be different for each time.

Because of this difference, the same job may get different estimated start times, future allocation, and reservation in different host partitions and queue-level scheduling. With time-based reservation configured, LSF always keeps the same reservation and future allocation with the earliest estimated start time.

## bjobs displays future allocation information

- ◆ By default, job future allocation contains LSF host list and number of CPUs per host, for example: `alloc=2*hostA 3*hostB`
- ◆ LSF integrations define their own future allocation string to override the default LSF allocation. For example, in RMS, future allocation is displayed as:  
`rms_alloc=2*sierra0 3*sierra1`

## Predicted start time may be postponed for some jobs

If a pending job cannot be placed in a future resource allocation, the scheduler can skip it in the start time reservation calculation and fall back to use greedy slot reservation. There are two possible reasons:

- ◆ The job slot request cannot be satisfied in the future allocation
- ◆ Other non-slot resources cannot be satisfied.

Either way, the scheduler continues calculating predicted start time for the remaining jobs without considering the skipped job.

Later, once the resource request of skipped job can be satisfied and placed in a future allocation, the scheduler reevaluates the predicted start time for the rest of jobs, which may potentially postpone their start times.

To minimize the overhead in recalculating the predicted start times to include previously skipped jobs, you should configure a small value for `LSB_TIME_RESERVE_NUMJOBS` in `lsf.conf`.

## Reservation scenarios

**Scenario 1** Even though no running jobs finish and no host status in cluster are changed, a job's future allocation may still change from time to time.

**Why this happens** Each scheduling cycle, the scheduler recalculates a job's reservation information, estimated start time and opportunity for future allocation. The job candidate host list may be reordered according to current load. This reordered candidate host list will be used for the entire scheduling cycle, also including job future allocation calculation. So different order of candidate hosts may lead to different result of job future allocation. However, the job estimated start time should be the same.

For example, there are two hosts in cluster, `hostA` and `hostB`. 4 CPUs per host. Job 1 is running and occupying 2 CPUs on `hostA` and 2 CPUs on `hostB`. Job 2 requests 6 CPUs. If the order of hosts is `hostA` and `hostB`, then the future allocation of job 2 will be 4 CPUs on `hostA` 2 CPUs on `hostB`. If the order of hosts changes in the next scheduling cycle changes to `hostB` and `hostA`, then the future allocation of job 2 will be 4 CPUs on `hostB` 2 CPUs on `hostA`.

**Scenario 2:** If you set `JOB_ACCEPT_INTERVAL` to non-zero value, after job is dispatched, within `JOB_ACCEPT_INTERVAL` period, pending job estimated start time and future allocation may momentarily fluctuate.

**Why this happens** The scheduler does a time-based reservation calculation each cycle. If `JOB_ACCEPT_INTERVAL` is set to non-zero value, once a new job has been dispatched to a host, this host will not accept new job within `JOB_ACCEPT_INTERVAL` interval. Because the host will not be considered for the entire scheduling cycle, no time-based reservation calculation is done, which may result in slight change in job estimated start time and future allocation information. After `JOB_ACCEPT_INTERVAL` has passed, host will become available for time-based reservation calculation again, and the pending job estimated start time and future allocation will be accurate again.

## Examples

**Example 1** Three hosts, 4 CPUs each: `qat24`, `qat25`, and `qat26`. Job 11895 uses 4 slots on `qat24` (10 hours). Job 11896 uses 4 slots on `qat25` (12 hours), and job 11897 uses 2 slots on `qat26` (9 hours).

Job 11898 is submitted and requests `-n 6 -R "span[ptile=2]"`.

```
bjobs -l 11898
Job <11898>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>,
 Job Priority <50>, Command <sleep 100000000>
..
RUNLIMIT
 840.0 min of hostA
Fri Apr 22 15:18:56: Reserved <2> job slots on host(s) <2*qat26>;
Sat Apr 23 03:28:46: Estimated Job Start Time;
 alloc=2*qat25 2*qat24 2*qat26.lsf.platform.com
```

**Example 2** Two RMS hosts, `sierraA` and `sierraB`, 8 CPUs per host. Job 3873 uses 4\*`sierra0` and will last for 10 hours. Job 3874 uses 4\*`sierra1` and will run for 12 hours. Job 3875 uses 2\*`sierra2` and 2\*`sierra3`, and will run for 13 hours.

Job 3876 is submitted and requests `-n 6 -ext "RMS[nodes=3]"`.

```
bjobs -l 3876
Job <3876>, User <user2>, Project <default>, Status <PEND>, Queue <rms>, Extsched <RMS[nodes=3]>, Command <sleep 1000000>
Fri Apr 22 15:35:28: Submitted from host <sierraa>, CWD <${HOME}>, 6 Processors Requested;

RUNLIMIT
840.0 min of sierraa
Fri Apr 22 15:35:46: Reserved <4> job slots on host(s) <4*sierrab>;
Sat Apr 23 01:34:12: Estimated job start time;
 rms_alloc=2*sierra[0,2-3]

...
```

**Example 3** Rerun example 1, but this time, use greedy slot reservation instead of time-based reservation:

```
bjobs -l 3876
Job <12103>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>, Job Priority <50>, Command <sleep 1000000>
Fri Apr 22 16:17:59: Submitted from host <qat26>, CWD <${HOME}>, 6 Processors Requested, Requested Resources <span[ptile=2]>;

RUNLIMIT
720.0 min of qat26
Fri Apr 22 16:18:09: Reserved <2> job slots on host(s) <2*qat26.lsf.platform.com>;
Sat Apr 23 01:39:13: Job will start no sooner than indicated time stamp;
```

## Viewing Resource Reservation Information

### View host-level resource information (bhosts)

- 1 Use `bhosts -l` to show the amount of resources reserved on each host. In the following example, 143 MB of memory is reserved on `hostA`, and no memory is currently available on the host.

```
bhosts -l hostA
HOST hostA
STATUS CPUF JL/U MAX NJOBS RUN SSUSP USUSP RSV DISPATCH_WINDOW
ok 20.00 - 4 2 1 0 0 1 -

CURRENT LOAD USED FOR SCHEDULING:
r15s r1m r15m ut pg io ls it tmp swp
mem
Total 1.5 1.2 2.0 91% 2.5 7 49 0 911M 915M
OM
Reserved 0.0 0.0 0.0 0% 0.0 0 0 0 0M 0M
143M
```

- 2 Use `bhosts -s` to view information about shared resources.

## Viewing Resource Reservation Information

### View queue-level resource information (bqueues)

- 1 Use `bqueues -l` to see the resource usage configured at the queue level.

```
bqueues -l reservation
QUEUE: reservation
 -- For resource reservation

PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
40 0 Open:Active - - - - 4 0 0 0 0 4

SCHEDULING PARAMETERS
loadSched r15s r1m r15m ut pg io ls it tmp swp mem
loadStop - - - - - - - - - - - -

 cpuspeed bandwidth
loadSched - -
loadStop - -

SCHEDULING POLICIES: RESOURCE_RESERVE

USERS: all users
HOSTS: all

Maximum resource reservation time: 600 seconds
```

### View reserved memory for pending jobs (bjobs)

If the job memory requirements cannot be satisfied, `bjobs -l` shows the pending reason. `bjobs -l` shows both reserved slots and reserved memory.

- 1 For example, the following job reserves 60 MB of memory on `hostA`:  
`bsub -m hostA -n 2 -q reservation -R"rusage[mem=60]" sleep 8888`  
Job <3> is submitted to queue <reservation>.  
`bjobs -l` shows the reserved memory:

```
bjobs -lp

Job <3>, User <user1>, Project <default>, Status <PEND>, Queue <reservation>
, Command <sleep 8888>
Tue Jan 22 17:01:05: Submitted from host <user1>, CWD </home/user1/>, 2 Processors
Requested, Requested Resources <rusage[mem=60]>, Specified Hosts <hostA>;
Tue Jan 22 17:01:15: Reserved <1> job slot on host <hostA>;
Tue Jan 22 17:01:15: Reserved <60> megabyte memory on host <60M*hostA>;
PENDING REASONS:
Not enough job slot(s): hostA;

SCHEDULING PARAMETERS
loadSched r15s r1m r15m ut pg io ls it tmp swp mem
loadStop - - - - - - - - - - - -
```



|           | cpuspeed | bandwidth |
|-----------|----------|-----------|
| loadSched | -        | -         |
| loadStop  | -        | -         |

---

## View per-resource reservation (bresources)

---

- 1 Use `bresources` to display per-resource reservation configurations from `lsb.resources`:

The following example displays all resource reservation configurations:

```
bresources -s
Begin ReservationUsage
RESOURCE METHOD
licenseX PER_JOB
licenseY PER_HOST
licenseZ PER_SLOT
End ReservationUsage
```

The following example displays only `licenseZ` configuration:

```
bresources -s licenseZ
RESOURCE METHOD
licenseZ PER_SLOT
```

---

## Viewing Resource Reservation Information

## Advance Reservation

### Contents

- ◆ [Understanding Advance Reservations](#) on page 404
- ◆ [Configure Advance Reservation](#) on page 405
- ◆ [Using Advance Reservation](#) on page 407

## Understanding Advance Reservations

Advance reservations ensure access to specific hosts during specified times. An advance reservation is essentially a lock on a number of processors.

Each reservation consists of the number of processors to reserve, a list of hosts for the reservation, a start time, an end time, and an owner. You can also specify a resource requirement string instead of or in addition to a list of hosts.

During the time the reservation is active, only users or groups associated with the reservation have access to start new jobs on the reserved hosts. The reservation is active only within the time frame specified, and any given host may have several reservations in place, some of which may be active at the same time.

When an advance reservation becomes active, LSF attempts to start all jobs that reference the reservation. By default, jobs that are already running on the hosts may continue, even though they do not reference the reservation. However, if a job that references a reservation is pending because the host has reached its job slot limit, LSF frees up a job slot on the host by suspending one of the jobs that does not reference the reservation. This is the only case where advance reservation overrides another LSF job scheduling policy.

Reservations can also be created for system maintenance. If a system reservation is active, no other jobs can use the reserved hosts, and LSF does not dispatch jobs to the specified hosts while the reservation is active.

Only LSF administrators or root can create or delete advance reservations. Any LSF user can view existing advance reservations.

LSF treats advance reservation like other deadlines, such as dispatch windows or run windows; LSF does not schedule jobs that are likely to be suspended when a reservation becomes active. Jobs referencing the reservation are killed when the reservation expires. LSF administrators can prevent running jobs from being killed when the reservation expires by changing the termination time of the job using the reservation (`bmod -t`) before the reservation window closes.

*Open* advance reservations allow jobs to run even after the associated reservation expires. A job with the open advance reservation will only be treated as an advance reservation job during the reservation window, after which it becomes a normal job. This prevents the job from being killed and ensures that LSF does not prevent any previously suspended jobs from running or interfere with any existing scheduling policies.

## Configure Advance Reservation

### Enable advance reservation

- 1 To enable advance reservation in your cluster, configure the advance reservation scheduling plugin `schmod_advrsv` in `lsb.modules`.

```
Begin PluginModule
SCH_PLUGIN RB_PLUGIN SCH_DISABLE_PHASES
schmod_default () ()
schmod_advrsv () ()
End PluginModule
```

### Advance reservation license

Advance reservation requires the `lsf_sched_advance_reservation` license feature in your license file and `LSF_Sched_Advance_Reservation` configured in the `PRODUCTS` line of `lsf.cluster.cluster_name`.

### Allow users to create advance reservations

By default, only LSF administrators or root can add or delete advance reservations. To allow other users to use `brsvadd` to create advance reservations and `brsvdel` to delete advance reservations, you need to configure advance reservation user policies.

- 1 Use the `ResourceReservation` section of `lsb.resources` to configure advance reservation policies for users.

A `ResourceReservation` section specifies:

- ◆ Users or user groups that can create reservations
- ◆ Hosts that can be used for the reservation
- ◆ Time window when reservations can be created

Each advance reservation policy is defined in a separate `ResourceReservation` section, so it is normal to have multiple `ResourceReservation` sections in `lsb.resources`.

Only `user1` and `user2` can make advance reservations on `hostA` and `hostB`. The reservation time window is between 8:00 a.m. and 6:00 p.m. every day:

```
Begin ResourceReservation
NAME = dayPolicy
USERS = user1 user2 # optional
HOSTS = hostA hostB # optional
TIME_WINDOW = 8:00-18:00 # weekly recurring reservation
End ResourceReservation
```

`user1` can add the following reservation for user `user2` to use on `hostA` every Friday between 9:00 a.m. and 11:00 a.m.:

```
brsvadd -m "hostA" -n 1 -u "user2" -t "5:9:0-5:11:0"
Reservation "user2#2" is created
```

## Configure Advance Reservation

Users can only delete reservations they created themselves. In the example, only user `user1` can delete the reservation; `user2` cannot. Administrators can delete any reservations created by users.

All users in user group `ugroup1` except `user1` can make advance reservations on any host in `hgroup1`, except `hostB`, between 10:00 p.m. and 6:00 a.m. every day:

```
Begin ResourceReservation
NAME = nightPolicy
USERS = ugroup1 ~user1
HOSTS = hgroup1 ~hostB
TIME_WINDOW = 20:00-8:00
End ResourceReservation
```

---

**IMPORTANT:** The not operator (`~`) does not exclude LSF administrators from the policy.

---

## USER\_ADVANCE\_RESERVATION is obsolete (lsb.params)

`USER_ADVANCE_RESERVATION` in `lsb.params` is obsolete in LSF Version 7. Use the `ResourceReservation` section configuration in `lsb.resources` to configure advance reservation policies for your cluster.

# Using Advance Reservation

## Advance reservation commands

Use the following commands to work with advance reservations:

|                      |                      |
|----------------------|----------------------|
| <code>brsvadd</code> | Add a reservation    |
| <code>brsvdel</code> | Delete a reservation |
| <code>brsvs</code>   | View reservations    |

## Add reservations

---

**NOTE:** By default, only LSF administrators or root can add or delete advance reservations.

---

- 1 Run `brsvadd` to create new advance reservations.

You must specify the following for the reservation:

- ◆ Number of processors to reserve—This number should be less than or equal to the actual number of CPUs for the hosts defined in the reservation.
- ◆ Hosts for the reservation
- ◆ Owners of the reservation
- ◆ Time period for the reservation: Either:
  - ❖ Begin time and end time for a one-time reservation, OR
  - ❖ Time window for a recurring reservation

The `brsvadd` command returns a reservation ID that you use when you submit a job that uses the reserved hosts. Any single user or user group can have a maximum of 100 reservation IDs.

---

## Specify hosts for the reservation

- 1 Use one or both of the following `brsvadd` options to specify hosts for which processors are reserved:
    - ❖ The `-m` option lists the hosts needed for the reservation. The hosts listed by the `-m` option can be local to the cluster or hosts leased from remote clusters. At job submission, LSF considers the hosts in the specified order. If you also specify a resource requirement string with the `-R` option, `-m` is optional.
    - ❖ The `-R` option selects hosts for the reservation according to a resource requirements string. Only hosts that satisfy the resource requirement expression are reserved. `-R` accepts any valid resource requirement string, but only the select string takes effect. If you also specify a host list with the `-m` option, `-R` is optional.
-

## Add a one-time reservation

- 1 Use the `-b` and `-e` options of `brsvadd` to specify the begin time and end time of a one-time advance reservation. One-time reservations are useful for dedicating hosts to a specific user or group for critical projects.

The day and time are in the form:

```
[[year:]month:]day:]hour:minute
```

with the following ranges:

- ◆ *year*: any year after 1900 (YYYY)
- ◆ *month*: 1-12 (MM)
- ◆ *day of the month*: 1-31 (dd)
- ◆ *hour*: 0-23 (hh)
- ◆ *minute*: 0-59 (mm)

You must specify at least `hour:minute`. Year, month, and day are optional.

Three fields are assumed to be `day:hour:minute`, four fields are assumed to be `month:day:hour:minute`, and five fields are `year:month:day:hour:minute`.

If you do not specify a day, LSF assumes the current day. If you do not specify a month, LSF assumes the current month. If you specify a year, you must specify a month.

You must specify a begin and an end time. The time value for `-b` must use the same syntax as the time value for `-e`. It must be earlier than the time value for `-e`, and cannot be earlier than the current time.

The following command creates a one-time advance reservation for 1024 processors on host `hostA` for user `user1` between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -n 1024 -m hostA -u user1 -b 6:0 -e 8:0
Reservation "user1#0" is created
```

The hosts specified by `-m` can be local to the cluster or hosts leased from remote clusters.

The following command creates a one-time advance reservation for 1024 processors on a host of any type for user `user1` between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0
Reservation "user1#1" is created
```

The following command creates a one-time advance reservation that reserves 12 CPUs on `hostA` between 6:00 p.m. on 01 December 2003 and 6:00 a.m. on 31 January 2004:

```
brsvadd -n 12 -m hostA -u user1 -b 2003:12:01:18:00 -e
2004:01:31:06:00
Reservation user1#2 is created
```



## Add a recurring reservation

- 1 Use the `-t` option of `brsvadd` to specify a recurring advance reservation. The `-t` option specifies a time window for the reservation. Recurring reservations are useful for scheduling regular system maintenance jobs.

The day and time are in the form:

```
[day:]hour[:minute]
```

with the following ranges:

- ◆ *day of the week*: 0-6
- ◆ *hour*: 0-23
- ◆ *minute*: 0-59

Specify a time window one of the following ways:

- ◆ *hour-hour*
- ◆ *hour:minute-hour:minute*
- ◆ *day:hour:minute-day:hour:minute*

You must specify at least the hour. Day of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (:00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

When the job starts running, the run limit of the reservation is set to the minimum of the job run limit (if specified), the queue run limit (if specified), or the duration of the reservation time window. LSF administrators can prevent running jobs from being killed when the reservation expires by changing the termination time of the job using the reservation (`bmod -t`) before the reservation window closes.

The following command creates an advance reservation for 1024 processors on two hosts `hostA` and `hostB` for user group `groupA` every Wednesday from 12:00 midnight to 3:00 a.m.:

```
brsvadd -n 2048 -m "hostA hostB" -g groupA -t "3:0:0-3:3:0"
Reservation "groupA#0" is created
```

The following command creates an advance reservation for 1024 processors on `hostA` for user `user2` every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -n 1024 -m "hostA" -u user2 -t "12:0-14:0"
Reservation "user2#0" is created
```

The following command creates a system reservation on `hostA` every Friday from 6:00 p.m. to 8:00 p.m.:

```
brsvadd -n 1024 -m hostA -s -t "5:18:0-5:20:0"
Reservation "system#0" is created
```

While the system reservation is active, no other jobs can use the reserved hosts, and LSF does not dispatch jobs to the specified hosts.

## Using Advance Reservation

The following command creates an advance reservation for 1024 processors on hosts `hostA` and `hostB` with more than 50 MB of swap space for user `user2` every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -n 1024 -R "swp > 50" -m "hostA hostB" -u user2 -t "12:0-14:0"
Reservation "user2#1" is created
```

## Add an open reservation

- 1 Use the `-o` option of `brsvadd` to create an open advance reservation. You must specify the same information as for normal advance reservations.

The following command creates a one-time open advance reservation for 1024 processors on a host of any type for user `user1` between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -o -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0
Reservation "user1#1" is created
```

The following command creates an open advance reservation for 1024 processors on `hostB` for user `user3` every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -o -n 1024 -m "hostB" -u user3 -t "12:0-14:0"
Reservation "user2#0" is created
```

## Remove an advance reservation

- 1 Use `brsvdel` to delete reservations. Specify the reservation ID for the reservation you want to delete. For example:

```
brsvdel user1#0
Reservation user1#0 is being deleted
```

You can delete more than one reservation at a time. Administrators can delete any reservation, but users may only delete their own reservations.

See Chapter 14, “[Time Syntax and Configuration](#)” for more information about specifying time windows in LSF.

## View reservations

- 1 Use `brsvs` to show current reservations:

```
brsvs
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
user1#0 user user1 1024 hostA:1024 11/12/6/0-11/12/8/0
user2#0 user user2 1024 hostA:1024 12:0-14:0 *
groupA#0 group groupA 2048 hostA:1024 3:0:0-3:3:0 *
 hostB:1024
system#0 sys system 1024 hostA:1024 5:18:0-5:20:0 *
```

In the `TIME_WINDOW` column:

- ◆ A one-time reservation displays fields separated by slashes (month/day/hour/minute). For example:  
11/12/14/0-11/12/18/0
- ◆ A recurring reservation displays fields separated by colons (day:hour:minute). An asterisk (\*) indicates a recurring reservation. For example:  
5:18:0-5:20:0 \*

## Show a weekly planner

- 1 Use `brsvs -p` to show a weekly planner for specified hosts using advance reservation. The `all` keyword shows the planner for all hosts with reservations. The output of `brsvs -p` is displayed in terms of weeks. The week starts on Sunday. The timeframe of a recurring reservation is not displayed, since it is unlimited. The timeframe of one-time reservation is displayed in terms of a week. If the reservation spans multiple weeks, these weeks are displayed separately. If a week contains a one-time reservation and a recurring reservation, the timeframe is displayed, since that is relevant for one-time reservation.

**TIP:** MXJ indicates the configured maximum number of job slots for the host (MXJ defined in `lsb.hosts`).

```
brsvs -p all
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
user1#0 user user1 1024 hostA:1024 11/12/6/0-11/12/8/0
user2#0 user user2 1024 hostA:1024 12:0-14:0 *
groupA#0 group groupA 2048 hostA:1024 3:0:0-3:3:0 *
 hostB:1024
system#0 sys system 1024 hostA:1024 5:18:0-5:20:0 *
```

```
HOST: hostA (MXJ = 1024)
Week: 11/11/2008 - 11/17/2008
Hour:Min Sun Mon Tue Wed Thu Fri Sat

0:0 0 0 0 1024 0 0 0
0:10 0 0 0 1024 0 0 0
0:20 0 0 0 1024 0 0 0
...
2:30 0 0 0 1024 0 0 0
2:40 0 0 0 1024 0 0 0
2:50 0 0 0 1024 0 0 0
3:0 0 0 0 0 0 0 0
3:10 0 0 0 0 0 0 0
3:20 0 0 0 0 0 0 0
...
5:30 0 0 0 0 0 0 0
5:40 0 0 0 0 0 0 0
5:50 0 0 0 0 0 0 0
6:0 0 1024 0 0 0 0 0
6:10 0 1024 0 0 0 0 0
6:20 0 1024 0 0 0 0 0
```

## Using Advance Reservation

```

...
7:30 0 1024 0 0 0 0 0
7:40 0 1024 0 0 0 0 0
7:50 0 1024 0 0 0 0 0
8:0 0 0 0 0 0 0 0
8:10 0 0 0 0 0 0 0
8:20 0 0 0 0 0 0 0
...
11:30 0 0 0 0 0 0 0
11:40 0 0 0 0 0 0 0
11:50 0 0 0 0 0 0 0
12:0 1024 1024 1024 1024 1024 1024 1024
12:10 1024 1024 1024 1024 1024 1024 1024
12:20 1024 1024 1024 1024 1024 1024 1024
...
13:30 1024 1024 1024 1024 1024 1024 1024
13:40 1024 1024 1024 1024 1024 1024 1024
13:50 1024 1024 1024 1024 1024 1024 1024
14:0 0 0 0 0 0 0 0
14:10 0 0 0 0 0 0 0
14:20 0 0 0 0 0 0 0
...
17:30 0 0 0 0 0 0 0
17:40 0 0 0 0 0 0 0
17:50 0 0 0 0 0 0 0
18:0 0 0 0 0 0 1024 0
18:10 0 0 0 0 0 1024 0
18:20 0 0 0 0 0 1024 0
...
19:30 0 0 0 0 0 1024 0
19:40 0 0 0 0 0 1024 0
19:50 0 0 0 0 0 1024 0
20:0 0 0 0 0 0 0 0
20:10 0 0 0 0 0 0 0
20:20 0 0 0 0 0 0 0
...
23:30 0 0 0 0 0 0 0
23:40 0 0 0 0 0 0 0
23:50 0 0 0 0 0 0 0

```

HOST: hostB (MXJ = 1024)

Week: 11/11/2008 - 11/17/2008

| Hour:Min | Sun | Mon | Tue | Wed  | Thu | Fri | Sat |
|----------|-----|-----|-----|------|-----|-----|-----|
| 0:0      | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| 0:10     | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| 0:20     | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| ...      |     |     |     |      |     |     |     |
| 2:30     | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| 2:40     | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| 2:50     | 0   | 0   | 0   | 1024 | 0   | 0   | 0   |
| 3:0      | 0   | 0   | 0   | 0    | 0   | 0   | 0   |
| 3:10     | 0   | 0   | 0   | 0    | 0   | 0   | 0   |
| 3:20     | 0   | 0   | 0   | 0    | 0   | 0   | 0   |
| ...      |     |     |     |      |     |     |     |

|       |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|
| 23:30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23:40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23:50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

---

## Show reservation types and associated jobs

---

- 1 Use the `-l` option of `brsvs` to show each advance reservation in long format.

The rows that follow the reservation information show the type of reservation (open or closed) and the status of any job associated with the specified reservation (EXIT, PEND, RUN, or SUSP).

```
brsvs -l
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
user1#0 user user1 1024 hostA:1024 11/12/6/0-11/12/8/0
```

```
Reservation Type: CLOSED
EXIT Jobs: 714
RUN Jobs: 715
```

```
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
user2#0 user user2 1024 hostA:1024 12:0-14:0 *
```

```
Reservation Type: OPEN
EXIT Jobs: 716
```

```
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
groupA#0 group groupA 2048 hostA:1024 3:0:0-3:3:0 *
 hostB:1024
```

```
Reservation Type: OPEN
EXIT Jobs: 717
PEND Jobs: 718 719
RUN Jobs: 720
```

```
RSVID TYPE USER NCPUS RSV_HOSTS TIME_WINDOW
system#0 sys system 1024 hostA:1024 5:18:0-5:20:0 *
```

```
Reservation Type: CLOSED
RUN Jobs: 721
SUSP Jobs: 722
```

---

## Show reservation ID

---

- 1 Use `bjobs -l` to show the reservation ID used by a job:

```
bjobs -l
Job <1152>, User <user1>, Project <default>, Status <PEND>, Queue
<normal>, Reservation <user1#0>, Command <myjob>
```

```
Mon Nov 12 5:13:21: Submitted from host <hostB>, CWD
</home/user1/jobs>;
```

---

## Submit and modify jobs using advance reservations

---

- 1 Use the `-U` option of `bsub` to submit jobs with a reservation ID. For example:

```
bsub -U user1#0 myjob
```

The job can only use hosts reserved by the reservation `user1#0`. By default, LSF selects only hosts in the reservation. Use the `-m` option to specify particular hosts within the list of hosts reserved by the reservation; you can only select from hosts that were included in the original reservation.

If you do not specify hosts (`bsub -m`) or resource requirements (`bsub -R`), the default resource requirement is to select hosts that are of any host type (LSF assumes `"type==any"` instead of `"type==local"` as the default select string).

If you later delete the advance reservation while it is still active, any pending jobs still keep the `"type==any"` attribute.

A job can only use one reservation. There is no restriction on the number of jobs that can be submitted to a reservation; however, the number of slots available on the hosts in the reservation may run out. For example, reservation `user2#0` reserves 1024 slots on `hostA`. When all 1024 slots on `hostA` are used by jobs referencing `user2#0`, `hostA` is no longer available to other jobs using reservation `user2#0`. Any single user or user group can have a maximum of 100 reservation IDs.

Jobs referencing the reservation are killed when the reservation expires. LSF administrators can prevent running jobs from being killed when the reservation expires by changing the termination time of the job using the reservation (`bmod -t`) before the reservation window closes.

---

## Modify job reservation ID

**Prerequisites:** You must be an administrator to perform this task.

---

- 1 Use the `-U` option of `bmod` to change a job to another reservation ID. For example:

```
bmod -U user1#0 1234
```

- 2 To cancel the reservation, use the `-Un` option of `bmod`. For example:

```
bmod -Un 1234
```

---

## Change job termination time

---

- 1 Before the reservation window closes, administrators can use the `-t` option of `bmod` to change the termination time of a running job that is using the reservation. This prevents the job from being killed when the reservation expires; it *does not* extend the actual reservation window. For example:

```
bmod -t 15:0 1234
```

LSF users and administrators can modify the termination time of an open advance reservation without interfering with current job scheduling policies. If the new termination time of the advance reservation window overlaps with other open advance reservations, the job will suspend at the time of the overlap. When all open advance reservations are closed, running jobs using the reservations will also suspend, and current scheduling policies will take effect for these jobs. For example, you specify two open advance reservations as follows:

```
brsvadd -o -n 1 -m hostA -u user1 -b 15:10 -e 15:30
Reservation user1#17 is created
brsvadd -o -n 1 -m hostA -u user1 -b 15:35 -e 15:50
Reservation user1#18 is created
```

At 15:15, while job 122 in user1#17 is running, you modify its termination time as follows:

```
bmod -t 15:40 122
```

This termination time overlaps the reservation window of user1#18, in which job 245 started at 15:35. After modifying the termination time of job 122, the following events occur:

| At this time ... | These events occur ...                                                                               |
|------------------|------------------------------------------------------------------------------------------------------|
| 15:35            | Job 122 in reservation user1#17 suspends<br>Job 245 in reservation user1#18 starts                   |
| 15:40            | Job 122 in reservation user#17 is still suspended<br>Job 245 in reservation user#18 continues to run |
| 15:50            | Job 122 in reservation user1#17 suspends<br>Job 245 in reservation user1#18 suspends                 |
| 15:51            | Job 122 in reservation user1#17 runs<br>Job 245 in reservation user1#18 suspends                     |

At 15:50, both Job 122 in reservation user#17 and Job 245 in reservation user#18 are suspended. At this point, the existing scheduling policies determine which job runs; in this example, Job 122 in reservation user#17 runs.

**IMPORTANT:** `bmod -t` will not change the termination time of a pending job.

## Job resource usage limits

A job using a reservation is subject to all job resource usage limits. If a limit is reached on a particular host in a reservation, jobs using that reservation cannot start on that host.

## Preemption and fairshare

Higher priority jobs can only preempt other jobs that use the same reservation. In fairshare, a lower priority job can run as long as no other higher priority share holders can access the same reservation as the lower priority job.

## Forcing a job to run before a reservation is active

LSF administrators can use `brun` to force jobs to run before the reservation is active, but the job must finish running before the time window of the reservation expires.

For example, if the administrator forces a job with a reservation to run one hour before the reservation is active, and the reservation period is 3 hours, a 4 hour run limit takes effect.

## Advance reservations across clusters

You can create and use advance reservation for the MultiCluster job forwarding model. To enable this feature, you must upgrade all clusters to LSF Version 7 or later.

See the *Using Platform LSF MultiCluster* for more information.

## View historical accounting information for advance reservations

- 1 Use the `-U` option of the `bacct` command to display accounting information about advance reservations.

`bacct -U` displays information similar to the `brsvs` command:

- ◆ The reservation ID specified on the `-U` option.
- ◆ The type of reservation: `user` or `system`
- ◆ The user names of users who used the `brsvadd` command to create the advance reservations
- ◆ The user names of the users who can use the advance reservations (with `bsub -U`)
- ◆ Number of CPUs reserved
- ◆ List of hosts for which processors are reserved
- ◆ Time window for the reservation.
  - ❖ A one-time reservation displays fields separated by slashes (month/day/hour/minute). For example:  
11/12/14/0-11/12/18/0
  - ◆ A recurring reservation displays fields separated by colons (day:hour:minute). For example:  
5:18:0 5:20:0

```
bacct -U user1#2
```

```
Accounting for:
```

- ```
- advanced reservation IDs: user1#2,
- advanced reservations created by user1,
```

```
-----
RSVID      TYPE      CREATOR   USER      NCPUS      RSV_HOSTS  TIME_WINDOW
user1#2    user      user1     user1     1          hostA:1    9/16/17/36-9/16/17/38
SUMMARY:
Total number of jobs:          4
Total CPU time consumed:      0.5 second
```


Maximum memory of a job: 4.2 MB
Maximum swap of a job: 5.2 MB
Total duration time: 0 hour 2 minute 0 second

Dispatch and Run Windows

Contents

- ◆ [Dispatch and Run Windows](#) on page 419
- ◆ [Run Windows](#) on page 419
- ◆ [Dispatch Windows](#) on page 420

Dispatch and Run Windows

Both dispatch and run windows are time windows that control when LSF jobs start and run.

- ◆ Dispatch windows can be defined in `lsb.hosts`. Dispatch and run windows can be defined in `lsb.queues`.
- ◆ Hosts can only have dispatch windows. Queues can have dispatch windows and run windows.
- ◆ Both windows affect job starting; only run windows affect the stopping of jobs.
- ◆ Dispatch windows define when hosts and queues are active and inactive. It does not control job submission.
- ◆ Run windows define when jobs can and cannot run. While a run window is closed, LSF cannot start any of the jobs placed in the queue, or finish any of the jobs already running.
- ◆ When a dispatch window closes, running jobs continue and finish, and no new jobs can be dispatched to the host or from the queue. When a run window closes, LSF suspends running jobs, but new jobs can still be submitted to the queue.

Run Windows

Queues can be configured with a run window, which specifies one or more time periods during which jobs in the queue are allowed to run. Once a run window is configured, jobs in the queue cannot run outside of the run window.

Jobs can be submitted to a queue at any time; if the run window is closed, the jobs remain pending until it opens again. If the run window is open, jobs are placed and dispatched as usual. When an open run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again.

Configure run windows

- 1 To configure a run window, set `RUN_WINDOW` in `lsb.queues`.

For example, to specify that the run window will be open from 4:30 a.m. to noon, type:

```
RUN_WINDOW = 4:30-12:00
```

You can specify multiple time windows.

For more information about the syntax of time windows, see [Specifying Time Windows](#) on page 269.

View information about run windows

- 1 Use `bqueues -l` to display information about queue run windows.

Dispatch Windows

Queues can be configured with a dispatch window, which specifies one or more time periods during which jobs are accepted. Hosts can be configured with a dispatch window, which specifies one or more time periods during which jobs are allowed to start.

Once a dispatch window is configured, LSF cannot dispatch jobs outside of the window. By default, no dispatch windows are configured (the windows are always open).

Dispatch windows have no effect on jobs that have already been dispatched to the execution host; jobs are allowed to run outside the dispatch windows, as long as the queue run window is open.

Queue-level

Each queue can have a dispatch window. A queue can only dispatch jobs when the window is open.

You can submit jobs to a queue at any time; if the queue dispatch window is closed, the jobs remain pending in the queue until the dispatch window opens again.

Host-level

Each host can have dispatch windows. A host is not eligible to accept jobs when its dispatch windows are closed.

Configure dispatch windows

Dispatch windows can be defined for both queues and hosts. The default is no restriction, or always open.

Configure host dispatch windows

- 1 To configure dispatch windows for a host, set `DISPATCH_WINDOW` in `lsb.hosts` and specify one or more time windows. If no host dispatch window is configured, the window is always open.

Configure queue dispatch windows

- 1 To configure dispatch windows for queues, set `DISPATCH_WINDOW` in `lsb.queues` and specify one or more time windows. If no queue dispatch window is configured, the window is always open.

Display queue dispatch windows

- 1 Use `bqueues -l` to display queue dispatch windows.

Display host dispatch windows

- 1 Use `bhosts -l` to display host dispatch windows.

Dispatch Windows

Job Dependencies

Contents

- ◆ [Job Dependency Scheduling](#) on page 423
- ◆ [Dependency Conditions](#) on page 424

Job Dependency Scheduling

About job dependency scheduling

Sometimes, whether a job should start depends on the result of another job. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed after the previous step finishes successfully, and all subsequent steps must be aborted if any step fails.

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

In LSF, any job can be dependent on other LSF jobs. When you submit a job, you use `bsub -w` to specify a dependency expression, usually based on the job states of preceding jobs.

LSF will not place your job unless this dependency expression evaluates to TRUE. If you specify a dependency on a job that LSF cannot find (such as a job that has not yet been submitted), your job submission fails.

Specify a job dependency

- 1 To specify job dependencies, use `bsub -w` to specify a dependency expression for the job.

Syntax

```
bsub -w 'dependency_expression'
```

The dependency expression is a logical expression composed of one or more dependency conditions. For syntax of individual dependency conditions, see [Dependency Conditions](#) on page 424.

- ◆ To make dependency expression of multiple conditions, use the following logical operators:
 - ❖ && (AND)
 - ❖ || (OR)
 - ❖ ! (NOT)
- ◆ Use parentheses to indicate the order of operations, if necessary.
- ◆ Enclose the dependency expression in single quotes (') to prevent the shell from interpreting special characters (space, any logic operator, or parentheses). If you use single quotes for the dependency expression, use double quotes for quoted items within it, such as job names.
- ◆ Job names specify only your own jobs, unless you are an LSF administrator.
- ◆ Use double quotes (") around job names that begin with a number.
- ◆ In Windows, enclose the dependency expression in double quotes (") when the expression contains a space. For example:
 - ❖ `bsub -w "exit(678, 0)"` requires double quotes in Windows.
 - ❖ `bsub -w 'exit(678,0)'` can use single quotes in Windows.
- ◆ In the job name, specify the wildcard character (*) at the end of a string, to indicate all jobs whose name begins with the string. For example, if you use `jobA*` as the job name, it specifies jobs named `jobA`, `jobA1`, `jobA_test`, `jobA.log`, etc.

Multiple jobs with the same name

By default, if you use the job name to specify a dependency condition, and more than one of your jobs has the same name, all of your jobs that have that name must satisfy the test.

To change this behavior, set `JOB_DEP_LAST_SUB` in `lsb.params` to 1. Then, if more than one of your jobs has the same name, the test is done on the one submitted most recently.

Dependency Conditions

The following dependency conditions can be used with any job:

- ◆ `done(job_ID | "job_name")`
- ◆ `ended(job_ID | "job_name")`
- ◆ `exit(job_ID [, [op] exit_code])`
- ◆ `exit("job_name" [, [op] exit_code])`
- ◆ `external(job_ID | "job_name" , "status_text")`
- ◆ `job_ID | "job_name"`
- ◆ `post_done(job_ID | "job_name")`
- ◆ `post_err(job_ID | "job_name")`

◆ `started(job_ID | "job_name")`

done

Syntax

`done(job_ID | "job_name")`

Description

The job state is DONE.

ended

Syntax

`ended(job_ID | "job_name")`

Description

The job state is EXIT or DONE.

exit

Syntax

`exit(job_ID | "job_name"[, [operator] exit_code])`

where *operator* represents one of the following relational operators:

>

>=

<

<=

==

!=

Description

The job state is EXIT, and the job's exit code satisfies the comparison test.

If you specify an exit code with no operator, the test is for equality (== is assumed).

If you specify only the job, any exit code satisfies the test.

Examples

```
exit (myjob)
```

The job named `myjob` is in the EXIT state, and it does not matter what its exit code was.

```
exit (678,0)
```

The job with job ID 678 is in the EXIT state, and terminated with exit code 0.

```
exit ("678",!=0)
```

The job named `678` is in the EXIT state, and terminated with any non-zero exit code.

external

Syntax

`external(job_ID | "job_name", "status_text")`

Specify the first word of the job status or message description (no spaces). Only the first word is evaluated.

Description

The job has the specified job status, or the text of the job's status begins with the specified word.

Dependency Conditions

Job ID or job name

Syntax `job_ID | "job_name"`

Description If you specify a job without a dependency condition, the test is for the DONE state (LSF assumes the “done” dependency condition by default).

post_done

Syntax `post_done(job_ID | "job_name")`

Description The job state is POST_DONE (the post-processing of specified job has completed without errors).

post_err

Syntax `post_err(job_ID | "job_name")`

Description The job state is POST_ERR (the post-processing of specified job has completed with errors).

started

Syntax `started(job_ID | "job_name")`

Description The job state is:

- ◆ RUN, DONE, or EXIT
- ◆ PEND or PSUSP, and the job has a pre-execution command (`bsub -E`) that is running

Advanced dependency conditions

Job arrays If you use job arrays, you can specify additional dependency conditions that only work with job arrays.

To use other dependency conditions with array jobs, specify elements of a job array in the usual way.

Job dependency examples

```
bsub -J "JobA" -w 'done(JobB)' command
```

The simplest kind of dependency expression consists of only one dependency condition. For example, if JobA depends on the successful completion of JobB, submit the job as shown.

```
-w 'done(312) && (started(Job2)|exit("99Job"))'
```

The submitted job will not start until the job with the job ID of 312 has completed successfully, and either the job named Job2 has started, or the job named 99Job has terminated abnormally.

```
-w "'210'"
```

The submitted job will not start unless the job named 210 is finished. The numeric job name should be doubly quoted, since the UNIX shell treats `-w "210"` the same as `-w 210`, which would evaluate the job with the job ID of 210.

Dependency Conditions

Job Priorities

Contents

- ◆ [User-Assigned Job Priority](#) on page 429
- ◆ [Automatic Job Priority Escalation](#) on page 431
- ◆ [Absolute Job Priority Scheduling](#) on page 431

User-Assigned Job Priority

User-assigned job priority provides controls that allow users to order their jobs in a queue. Job order is the first consideration to determine job eligibility for dispatch. Jobs are still subject to all scheduling policies regardless of job priority. Jobs with the same priority are ordered first come first served.

The job owner can change the priority of their own jobs. LSF and queue administrators can change the priority of all jobs in a queue.

User-assigned job priority is enabled for all queues in your cluster, and can be configured with automatic job priority escalation to automatically increase the priority of jobs that have been pending for a specified period of time.

Considerations

The `btop` and `bbot` commands move jobs relative to other jobs of the same priority. These commands do not change job priority.

In this section

- ◆ [Configure job priority](#) on page 430
- ◆ [Specify job priority](#) on page 430
- ◆ [View job priority information](#) on page 430

Configure job priority

- 1 To configure user-assigned job priority edit `lsb.params` and define `MAX_USER_PRIORITY`. This configuration applies to all queues in your cluster.
- 2 Use `bparams -l` to display the value of `MAX_USER_PRIORITY`.

Syntax

```
MAX_USER_PRIORITY=max_priority
```

Where:

max_priority

Specifies the maximum priority a user can assign to a job. Valid values are positive integers. Larger values represent higher priority; 1 is the lowest.

LSF and queue administrators can assign priority beyond *max_priority*.

Example

```
MAX_USER_PRIORITY=100
```

Specifies that 100 is the maximum job priority that can be specified by a user.

Specify job priority

- ◆ Job priority is specified at submission using `bsub` and modified after submission using `bmod`. Jobs submitted without a priority are assigned the default priority of `MAX_USER_PRIORITY/2`.

Syntax

```
bsub -sp priority  
bmod [-sp priority | -spn] job_ID
```

Where:

-sp *priority*

Specifies the job priority. Valid values for *priority* are any integers between 1 and `MAX_USER_PRIORITY` (displayed by `bparams -l`). Incorrect job priorities are rejected.

LSF and queue administrators can specify priorities beyond `MAX_USER_PRIORITY`.

-spn

Sets the job priority to the default priority of `MAX_USER_PRIORITY/2` (displayed by `bparams -l`).

View job priority information

- 1 Use the following commands to view job history, the current status and system configurations:

`bhist -l job_ID`

Displays the history of a job including changes in job priority.

bjobs -l [job_ID] Displays the current job priority and the job priority at submission time. Job priorities are changed by the job owner, LSF and queue administrators, and automatically when automatic job priority escalation is enabled.

bparams -l Displays values for:

- ◆ The maximum user priority, `MAX_USER_PRIORITY`
- ◆ The default submission priority, `MAX_USER_PRIORITY/2`
- ◆ The value and frequency used for automatic job priority escalation, `JOB_PRIORITY_OVER_TIME`

Automatic Job Priority Escalation

Automatic job priority escalation automatically increases job priority of jobs that have been pending for a specified period of time. User-assigned job priority (see [User-Assigned Job Priority](#) on page 429) must also be configured.

As long as a job remains pending, LSF will automatically increase the job priority beyond the maximum priority specified by `MAX_USER_PRIORITY`. Job priority will not be increased beyond the value of `max_int` on your system.

Configure job priority escalation

- 1 To configure job priority escalation edit `lsb.params` and define `JOB_PRIORITY_OVER_TIME`.
User-assigned job priority must also be configured,
- 2 Use `bparams -l` to display the values of `JOB_PRIORITY_OVER_TIME`.

Syntax

```
JOB_PRIORITY_OVER_TIME=increment/interval
```

Where:

increment

Specifies the value used to increase job priority every *interval* minutes. Valid values are positive integers.

interval

Specifies the frequency, in minutes, to *increment* job priority. Valid values are positive integers.

Example

```
JOB_PRIORITY_OVER_TIME=3/20
```

Specifies that every 20 minute *interval* *increment* to job priority of pending jobs by 3.

Absolute Job Priority Scheduling

Absolute job priority scheduling (APS) provides a mechanism to control the job dispatch order to prevent job starvation.

When configured in a queue, APS sorts pending jobs for dispatch according to a job priority value calculated based on several configurable job-related factors. Each job priority weighting factor can contain subfactors. Factors and subfactors can be independently assigned a weight.

IMPORTANT: APS requires a Platform_HPC license.

APS provides administrators with detailed yet straightforward control of the job selection process.

- ◆ APS only sorts the jobs; job scheduling is still based on configured LSF scheduling policies. LSF attempts to schedule and dispatch jobs based on their order in the APS queue, but the dispatch order is not guaranteed.
- ◆ The job priority is calculated for pending jobs across multiple queues based on the sum of configurable factor values. Jobs are then ordered based on the calculated APS value.
- ◆ You can adjust the following for APS factors:
 - ❖ A weight for scaling each job-related factor and subfactor
 - ❖ Limits for each job-related factor and subfactor
 - ❖ A grace period for each factor and subfactor
- ◆ Administrators can configure absolute priority scheduling (APS) across multiple queues in APS queue groups. A queue group uses a single formula to calculate APS values. Jobs submitted to queues in the queue group are dispatched based on the priority of the master queue in the group.
- ◆ Administrators can also set a static system APS value for a job. A job with a system APS priority is guaranteed to have a higher priority than any calculated value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.
- ◆ Administrators can use the ADMIN factor to manually adjust the calculated APS value for individual jobs.

Scheduling priority factors

To calculate the job priority, APS divides job-related information into several categories. Each category becomes a factor in the calculation of the scheduling priority. You can configure the weight, limit, and grace period of each factor to get the desired job dispatch order.

LSF will sum the value of each factor based on the weight of each factor.

Factor weight The weight of a factor expresses the importance of the factor in the absolute scheduling priority. The factor weight is multiplied by the value of the factor to change the factor value. A positive weight increases the importance of the factor, and a negative weight decreases the importance of a factor. Undefined factors have a weight of 0, which causes the factor to be ignored in the APS calculation.

Factor limit The limit of a factor sets the minimum and maximum absolute value of each weighted factor. Factor limits must be positive values.

Factor grace period Each factor can be configured with a grace period. The factor will only be counted as part of the APS value when the job has been pending for a long time and it exceeds the grace period.

Factors and subfactors

Factors	Subfactors	Metric
FS (user based fairshare factor)	The existing fairshare feature tunes the dynamic user priority	<p>The fairshare factor automatically adjusts the APS value based on dynamic user priority.</p> <p>FAIRSHARE must be defined in the queue. The FS factor is ignored for non-fairshare queues.</p> <p>The FS factor is influenced by the following fairshare parameters in <code>lsb.params</code>:</p> <ul style="list-style-type: none"> ◆ CPU_TIME_FACTOR ◆ RUN_TIME_FACTOR ◆ RUN_JOB_FACTOR ◆ HIST_HOURS
RSRC (resource factors)	PROC	Requested processors will be the max of <code>bsub -n min,max</code> , the min of <code>bsub -n min</code> , or the value of <code>PROCLIMIT</code> in <code>lsb.queues</code> .
	MEM	Total real memory requested (in MB). Memory requests appearing to the right of a <code> </code> symbol in a usage string are ignored in the APS calculation.
	SWAP	Total swap space requested (in MB). As with MEM, swap space requests appearing to the right of a <code> </code> symbol in a usage string are ignored.

Factors	Subfactors	Metric
WORK (job attributes)	JRIORITY	The job priority specified by: <ul style="list-style-type: none"> ◆ Default specified by MAX_USER_PRIORITY in <code>lsb.params</code> ◆ Users with <code>bsub -sp</code> or <code>bmod -sp</code> ◆ Automatic priority escalation with JOB_PRIORITY_OVER_TIME in <code>lsb.params</code>
	QRIORITY	The priority of the submission queue.
ADMIN		Administrators use <code>bmod -aps</code> to set this subfactor value for each job. A positive value increases the APS. A negative value decreases the APS. The ADMIN factor is added to the calculated APS value to change the factor value. The ADMIN factor applies to the entire job. You cannot configure separate weight, limit, or grace period factors. The ADMIN factor takes effect as soon as it is set.

Where LSF gets the job information for each factor

Factor or subfactor	Gets job information from ...
MEM	The value for jobs submitted with <code>-R "rusage[mem]"</code>
SWAP	The value for jobs submitted with <code>-R "rusage[swp]"</code>
PROC	The value of <i>n</i> for jobs submitted with <code>bsub -n (min, max)</code> , or the value of PROCLIMIT in <code>lsb.queues</code>
JRIORITY	The dynamic priority of the job, updated every scheduling cycle and escalated by interval defined in JOB_PRIORITY_OVER_TIME defined in <code>lsb.params</code>
QRIORITY	The priority of the job submission queue
FS	The fairshare priority value of the submission user

Enable absolute priority scheduling

Configure APS_PRIORITY in an absolute priority queue in `lsb.queues`.

```
APS_PRIORITY=WEIGHT[[factor, value] [subfactor, value]...]...] LIMIT[[factor, value] [subfactor, value]...]...] GRACE_PERIOD[[factor, value] [subfactor, value]...]...]
```

Pending jobs in the queue will be ordered according to the calculated APS value. If weight of a subfactor is defined, but the weight of parent factor is not defined, the parent factor weight will be set as 1.

The WEIGHT and LIMIT factors are floating-point values. Specify a *value* for GRACE_PERIOD in seconds (*values*), minutes (*valuem*), or hours (*valueh*).

The default unit for grace period is hours.

For example, the following sets a grace period of 10 hours for the MEM factor, 10 minutes for the JPRIORITY factor, 10 seconds for the QPRIORITY factor, and 10 hours (default) for the RSRC factor:

```
GRACE_PERIOD[[MEM,10h] [JPRIORITY, 10m] [QPRIORITY,10s] [RSRC, 10]]
```

You cannot specify zero (0) for the WEIGHT, LIMIT, and GRACE_PERIOD of any factor or subfactor.

APS queues cannot configure cross-queue fairshare (FAIRSHARE_QUEUES) or host-partition fairshare.

Modify the system APS value (bmod)

The absolute scheduling priority for a newly submitted job is dynamic. Job priority is calculated and updated based on formula specified by APS_PRIORITY in the absolute priority queue. Administrators can use `bmod` to manually override the calculated APS value.

Run `bmod -apsn job_ID` to undo the previous `bmod -aps` setting.

Assign a static system priority and ADMIN factor value

Administrators can use using `bmod -aps "system=value"` to assign a static job priority for a pending job. The value cannot be zero (0).

In this case, job's absolute priority will not be calculated. The system APS priority is guaranteed to be higher than any calculated APS priority value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.

The system APS value set by `bmod -aps` is preserved after `mbatchd` reconfiguration or `mbatchd` restart.

Use the ADMIN factor to adjust the APS value

Administrators can use `bmod -aps "admin=value"` to change the calculated APS value for a pending job. The ADMIN factor is added to the calculated APS value to change the factor value. The absolute priority of the job is recalculated. The value cannot be zero (0).

A `bmod -aps` command always overrides the last `bmod -aps` commands

The ADMIN APS value set by `bmod -aps` is preserved after `mbatchd` reconfiguration or `mbatchd` restart.

Example bmod output

The following commands change the APS values for jobs 313 and 314:

```
bmod -aps "system=10" 313
```

```
Parameters of job <313> are being changed
```

```
bmod -aps "admin=10.00" 314
```

```
Parameters of job <314> are being changed
```

View modified APS values

Use `bjobs -aps` to see the effect of the changes:

```
bjobs -aps
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME	APS
313	user1	PEND	owners	hostA		myjob	Feb 12 01:09	(10)
321	user1	PEND	owners	hostA		myjob	Feb 12 01:09	-
314	user1	PEND	normal	hostA		myjob	Feb 12 01:08	109.00

Absolute Job Priority Scheduling

```
312    user1    PEND    normal hostA                myjob Feb 12 01:08 99.00
315    user1    PEND    normal hostA                myjob Feb 12 01:08 99.00
316    user1    PEND    normal hostA                myjob Feb 12 01:08 99.00
```

Use `bjobs -l` to show APS values modified by the administrator:

```
bjobs -l
```

```
Job <313>, User <user1>, Project <default>, Service Class
<SLASamples>, Status <RUN>, Queue <normal>, Command <myjob>, System
Absolute Priority <10>
```

```
Job <314>, User <user1>, Project <default>, Status <PEND>, Queue
<normal>, Command <myjob>, Admin factor value <10>
```

Use `bhist -l` to see historical information about administrator changes to APS values. For example, after running these commands:

```
1 bmod -aps "system=10" 108
```

```
2 bmod -aps "admin=20" 108
```

```
3 bmod -apsn 108
```

`bhist -l` shows the sequence changes to job 108:

```
bhist -l
```

```
Job <108>, User <user1>, Project <default>, Command <sleep 10000>
```

```
Tue Feb 13 15:15:26: Submitted from host <HostB>, to Queue <normal>, CWD
</scratch/user1>;
```

```
Tue Feb 13 15:15:40: Parameters of Job are changed:
```

```
  Absolute Priority Scheduling factor string changed to : system=10;
```

```
Tue Feb 13 15:15:48: Parameters of Job are changed:
```

```
  Absolute Priority Scheduling factor string changed to : admin=20;
```

```
Tue Feb 13 15:15:58: Parameters of Job are changed:
```

```
  Absolute Priority Scheduling factor string deleted;
```

```
Summary of time in seconds spent in various states by Tue Feb 13 15:16:02
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
36	0	0	0	0	0	36

Configure APS across multiple queues

Use `QUEUE_GROUP` in an absolute priority queue in `lsb.queues` to configure APS across multiple queues.

When APS is enabled in the queue with `APS_PRIORITY`, the `FAIRSHARE_QUEUES` parameter is ignored. The `QUEUE_GROUP` parameter replaces `FAIRSHARE_QUEUES`, which is obsolete in LSF 7.0.

Example 1

You want to schedule jobs from the normal queue and the short queue, factoring the job priority (weight 1) and queue priority (weight 10) in the APS value:

```
Begin Queue
```

```
QUEUE_NAME    = normal
```

```
PRIORITY      = 30
```

```
NICE          = 20
```

```
APS_PRIORITY  = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10]]
```

```
QUEUE_GROUP   = short
```

```
DESCRIPTION = For normal low priority jobs, running only if hosts
are lightly loaded.
End Queue
```

```
...
```

```
Begin Queue
QUEUE_NAME = short
PRIORITY = 20
NICE = 20
End Queue
```

The APS value for jobs from the normal queue and the short queue will be: calculated as:

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority})$$

The first 1 is the weight of the WORK factor; the second 1 is the weight of the job priority subfactor; the 10 is the weight of queue priority subfactor.

If you want the job priority to increase based on the pending time, you must configure `JOB_PRIORITY_OVER_TIME` parameter in the `lsb.params`.

Example 2

Extending example 1, you want to add user-based fairshare with a weight of 100 to the APS value in the normal queue:

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
NICE = 20
FAIRSHARE = USER_SHARES [[user1, 5000] [user2, 5000] [others, 1]]
APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10] [FS, 100]]
QUEUE_GROUP = short
DESCRIPTION = For normal low priority jobs, running only if hosts
are lightly loaded.
End Queue
```

The APS value will now be calculated as

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority}) + 100 * \text{user_priority}$$

Example 3

Extending example 2, you now to add swap space to the APS value calculation. The APS configuration changes to:

```
APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10] [FS, 100] [SWAP, -10]]
```

And the APS value is now calculated as

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority}) + 100 * \text{user_priority} + 1 * (-10 * \text{SWAP})$$

View pending job order by the APS value

Run `bjobs -aps` to see APS information for pending jobs in the order of absolute scheduling priority. The order that the pending jobs are displayed is the order in which the jobs will be considered for dispatch.

The APS value is calculated based on the current scheduling cycle, so jobs are not guaranteed to be dispatched in this order.

Pending jobs are ordered by APS value. Jobs with system APS values are listed first, from highest to lowest APS value. Jobs with calculated APS values are listed next ordered from high to low value. Finally, jobs not in an APS queue are listed. Jobs with equal APS values are listed in order of submission time.

If queues are configured with the same priority, `bjobs -aps` may not show jobs in the correct expected dispatch order. Jobs may be dispatched in the order the queues are configured in `lsb.queues`. You should avoid configuring queues with the same priority.

Example `bjobs -aps` output

The following example uses this configuration;

- ◆ The APS only considers the job priority and queue priority for jobs from normal queue (priority 30) and short queue (priority 20)
 - ❖ `APS_PRIORITY = WEIGHT [[QRIORITY, 10] [JPRIORITY, 1]]`
 - ❖ `QUEUE_GROUP = short`
- ◆ Priority queue (40) and idle queue (15) do not use APS to order jobs
- ◆ `JOB_PRIORITY_OVER_TIME=5/10` in `lsb.params`
- ◆ `MAX_USER_PRIORITY=100` in `lsb.params`

`bjobs -aps` was run at 14:41:

`bjobs -aps`

JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME	APS
15	User2	PEND	priority	HostB	myjob	Dec 21 14:30	-
22	User1	PEND	Short	HostA	myjob	Dec 21 14:30	(60)
2	User1	PEND	Short	HostA	myjob	Dec 21 11:00	360
12	User2	PEND	normal	HostB	myjob	Dec 21 14:30	355
4	User1	PEND	Short	HostA	myjob	Dec 21 14:00	270
5	User1	PEND	Idle	HostA	myjob	Dec 21 14:01	-

For job 2, $APS = 10 * 20 + 1 * (50 + 220 * 5 / 10) = 360$

For job 12, $APS = 10 * 30 + 1 * (50 + 10 * 5 / 10) = 355$

For job 4, $APS = 10 * 20 + 1 * (50 + 40 * 5 / 10) = 270$

View APS configuration for a queue

Run `bqueues -l` to see the current APS information for a queue:

`bqueues -l normal`

QUEUE: normal

-- No description provided. This is the default queue.

PARAMETERS/STATISTICS

PRIO	NICE	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
500	20	Open:Active	-	-	-	-	0	0	0	0	0	0

SCHEDULING PARAMETERS

r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem

```
loadSched - - - - - - - - - -
loadStop  - - - - - - - - - -
```

SCHEDULING POLICIES: FAIRSHARE APS_PRIORITY

APS_PRIORITY:

	WEIGHT FACTORS	LIMIT FACTORS	GRACE PERIOD
FAIRSHARE	10000.00	-	-
RESOURCE	101010.00	-	1010h
PROCESSORS	-10.01	-	-
MEMORY	1000.00	20010.00	3h
SWAP	10111.00	-	-
WORK	1.00	-	-
JOB PRIORITY	-999999.00	10000.00	4131s
QUEUE PRIORITY	10000.00	10.00	-

USER_SHARES: [user1, 10]

SHARE_INFO_FOR: normal/

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user1	10	3.333	0	0	0.0	0

USERS: all

HOSTS: all

REQUEUE_EXIT_VALUES: 10

Feature interactions

Fairshare

The default user-based fairshare can be a factor in APS calculation by adding the FS factor to APS_PRIORITY in the queue.

- ◆ APS cannot be used together with DISPATCH_ORDER=QUEUE.
- ◆ APS cannot be used together with cross-queue fairshare (FAIRSHARE_QUEUES). The QUEUE_GROUP parameter replaces FAIRSHARE_QUEUES, which is obsolete in LSF 7.0.
- ◆ APS cannot be used together with queue-level fairshare or host-partition fairshare.

FCFS

APS overrides the job sort result of FCFS.

SLA scheduling

APS cannot be used together with SLA scheduling.

Job requeue

All requeued jobs are treated as newly submitted jobs for APS calculation. The job priority, system, and ADMIN APS factors are reset on requeue.

Rerun jobs

Rerun jobs are not treated the same as requeued jobs. A job typically reruns because the host failed, not through some user action (like job requeue), so the job priority is not reset for rerun jobs.

Job migration	<p>Suspended (<code>bstop</code>) jobs and migrated jobs (<code>bmig</code>) are always scheduled before pending jobs. For migrated jobs, LSF keeps the existing job priority information. If <code>LSB_REQUEUE_TO_BOTTOM</code> and <code>LSB_MIG2PEND</code> are configured in <code>lsf.conf</code>, the migrated jobs keep their APS information. When <code>LSB_REQUEUE_TO_BOTTOM</code> and <code>LSB_MIG2PEND</code> are configured, the migrated jobs need to compete with other pending jobs based on the APS value. If you want to reset the APS value, the you should use <code>brequeue</code>, not <code>bmig</code>.</p>
Resource reservation	<p>The resource reservation is based on queue policies. The APS value will not affect current resource reservation policy.</p>
Preemption	<p>The preemption is based on queue policies. The APS value will not affect the current preemption policy.</p>
Chunk jobs	<p>The first chunk job to be dispatched will be picked based on the APS priority. Other jobs in the chunk will be picked based on the APS priority and the default chunk job scheduling policies.</p> <p>The following job properties must be the same for all chunk jobs:</p> <ul style="list-style-type: none">◆ Submitting user◆ Resource requirements◆ Host requirements◆ Queue or application profile◆ Job priority
Backfill scheduling	<p>Not affected.</p>
Advance reservation	<p>Not affected.</p>

Job Requeue and Job Rerun

Contents

- ◆ [About Job Requeue](#) on page 441
- ◆ [Automatic Job Requeue](#) on page 441
- ◆ [Reverse Requeue](#) on page 442
- ◆ [Exclusive Job Requeue](#) on page 442
- ◆ [User-Specified Job Requeue](#) on page 443
- ◆ [Automatic Job Rerun](#) on page 444

About Job Requeue

A networked computing environment is vulnerable to any failure or temporary conditions in network services or processor resources. For example, you might get NFS stale handle errors, disk full errors, process table full errors, or network connectivity problems. Your application can also be subject to external conditions such as a software license problems, or an occasional failure due to a bug in your application.

Such errors are temporary and probably will happen at one time but not another, or on one host but not another. You might be upset to learn all your jobs exited due to temporary errors and you did not know about it until 12 hours later.

LSF provides a way to automatically recover from temporary errors. You can configure certain exit values such that in case a job exits with one of the values, the job will be automatically requeued as if it had not yet been dispatched. This job will then be retried later. It is also possible for you to configure your queue such that a requeued job will not be scheduled to hosts on which the job had previously failed to run.

Automatic Job Requeue

You can configure a queue to automatically requeue a job if it exits with a specified exit value.

- ◆ The job is requeued to the head of the queue from which it was dispatched, unless the `LSB_REQUEUE_TO_BOTTOM` parameter in `lsf.conf` is set.

Reverse Requeue

- ◆ When a job is requeued, LSF does not save the output from the failed run.
- ◆ When a job is requeued, LSF does not notify the user by sending mail.
- ◆ A job terminated by a signal is not requeued.

Configure automatic job requeue

- 1 To configure automatic job requeue, set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queues`) and specify the exit codes that will cause the job to be requeued.

```
Begin Queue
...
REQUEUE_EXIT_VALUES = 99 100
...
End Queue
```

This configuration enables jobs that exit with 99 or 100 to be requeued.

Reverse Requeue

By default, if you use automatic job requeue, jobs are requeued to the head of a queue. You can have jobs requeued to the bottom of a queue instead. The job priority does not change.

Configure reverse requeue

You must already use automatic job requeue (`REQUEUE_EXIT_VALUES` in `lsb.queues`).

To configure reverse requeue:

- 1 Set `LSB_REQUEUE_TO_BOTTOM` in `lsf.conf` to 1.
- 2 Reconfigure the cluster:
 - a `lsadmin reconfig`
 - b `badmin mbdrestart`

Exclusive Job Requeue

You can configure automatic job requeue so that a failed job is not rerun on the same host.

Limitations

- ◆ If `mbatchd` is restarted, this feature might not work properly, since LSF forgets which hosts have been excluded. If a job ran on a host and exited with an exclusive exit code before `mbatchd` was restarted, the job could be dispatched to the same host again after `mbatchd` is restarted.
- ◆ Exclusive job requeue does not work for MultiCluster jobs or parallel jobs
- ◆ A job terminated by a signal is not requeued

Configure exclusive job requeue

- 1 Set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queues`) and define the exit code using parentheses and the keyword `EXCLUDE`, as shown:
`EXCLUDE(exit_code...)`

When a job exits with any of the specified exit codes, it will be requeued, but it will not be dispatched to the same host again.

```
Begin Queue
...
REQUEUE_EXIT_VALUES=30 EXCLUDE(20)
HOSTS=hostA hostB hostC
...
End Queue
```

A job in this queue can be dispatched to `hostA`, `hostB` or `hostC`.

If a job running on `hostA` exits with value 30 and is requeued, it can be dispatched to `hostA`, `hostB`, or `hostC`. However, if a job running on `hostA` exits with value 20 and is requeued, it can only be dispatched to `hostB` or `hostC`.

If the job runs on `hostB` and exits with a value of 20 again, it can only be dispatched on `hostC`. Finally, if the job runs on `hostC` and exits with a value of 20, it cannot be dispatched to any of the hosts, so it will be pending forever.

User-Specified Job Requeue

You can use `brequeue` to kill a job and requeue it. When the job is requeued, it is assigned the `PEND` status and the job's new position in the queue is after other jobs of the same priority.

Requeue a job

- 1 To requeue one job, use `brequeue`.
 - ◆ You can only use `brequeue` on running (`RUN`), user-suspended (`USUSP`), or system-suspended (`SSUSP`) jobs.
 - ◆ Users can only requeue their own jobs. Only root and LSF administrator can requeue jobs submitted by other users.
 - ◆ You cannot use `brequeue` on interactive batch jobs

```
brequeue 109
```

LSF kills the job with job ID 109, and requeues it in the `PEND` state. If job 109 has a priority of 4, it is placed after all the other jobs with the same priority.

```
brequeue -u User5 45 67 90
```

LSF kills and requeues 3 jobs belonging to `User5`. The jobs have the job IDs 45, 67, and 90.

Automatic Job Rerun

Job requeue vs. job rerun

Automatic job requeue occurs when a job finishes and has a specified exit code (usually indicating some type of failure).

Automatic job rerun occurs when the execution host becomes unavailable while a job is running. It does not occur if the job itself fails.

About job rerun

When a job is rerun or restarted, it is first returned to the queue from which it was dispatched with the same options as the original job. The priority of the job is set sufficiently high to ensure the job gets dispatched before other jobs in the queue. The job uses the same job ID number. It is executed when a suitable host is available, and an email message is sent to the job owner informing the user of the restart.

Automatic job rerun can be enabled at the job level, by the user, or at the queue level, by the LSF administrator. If automatic job rerun is enabled, the following conditions cause LSF to rerun the job:

- ◆ The execution host becomes unavailable while a job is running
- ◆ The system fails while a job is running

When LSF reruns a job, it returns the job to the submission queue, with the same job ID. LSF dispatches the job as if it was a new submission, even if the job has been checkpointed.

Execution host fails

If the execution host fails, LSF dispatches the job to another host. You receive a mail message informing you of the host failure and the requeuing of the job.

LSF system fails

If the LSF system fails, LSF requeues the job when the system restarts.

Configure queue-level job rerun

-
- 1 To enable automatic job rerun at the queue level, set `RERUNNABLE` in `lsb.queues` to `yes`.
-

Submit a rerunnable job

-
- 1 To enable automatic job rerun at the job level, use `bsub -r`.
Interactive batch jobs (`bsub -I`) cannot be rerunnable.
-

Submit a job as not rerunnable

- 1 To disable automatic job rerun at the job level, use `bsub -rn`.

Disable post-execution for rerunnable jobs

Running of post-execution commands upon restart of a rerunnable job may not always be desirable; for example, if the post-exec removes certain files, or does other cleanup that should only happen if the job finishes successfully.

- 1 Use `LSB_DISABLE_RERUN_POST_EXEC=Y` in `lsf.conf` to prevent the post-exec from running when a job is rerun.

Automatic Job Rerun

Job Checkpoint, Restart, and Migration

Job checkpoint and restart optimizes resource usage by enabling a non-interactive job to restart on a new host from the point at which the job stopped—checkpointed jobs do not have to restart from the beginning. Job migration facilitates load balancing by enabling users to move a job from one host to another while taking advantage of job checkpoint and restart functionality.

Contents

- ◆ [Checkpoint and restart options](#) on page 447
- ◆ [Checkpoint directory and files](#) on page 448
- ◆ [Checkpoint and restart executables](#) on page 449
- ◆ [Job restart](#) on page 449
- ◆ [Job migration](#) on page 450

Checkpoint and restart options

You can implement job checkpoint and restart at one of the following levels.

- ◆ Kernel level—provided by your operating system, enabled by default
- ◆ User level—provided by special LSF libraries that you link to your application object files
- ◆ Application level—provided by your site-specific applications and supported by LSF through the use of application-specific `echkpt` and `erestart` executables

NOTE: For a detailed description of the job checkpoint and restart feature and how to configure it, see the *Platform LSF Configuration Reference*.

Checkpoint directory and files

The job checkpoint and restart feature requires that a job be made checkpointable at the job or queue level. LSF users can make a job checkpointable by submitting the job using `bsub -k` and specifying a checkpoint directory. Queue administrators can make all jobs in a queue checkpointable by specifying a checkpoint directory for the queue.

Requirements

The following requirements apply to a checkpoint directory specified at the queue level:

- ◆ The specified checkpoint directory must already exist. LSF will not create the checkpoint directory.
- ◆ The user account that submits the job must have read and write permissions for the checkpoint directory.
- ◆ For the job to restart on another execution host, both the original and new hosts must have network connectivity to the checkpoint directory.

Behavior

Specifying a checkpoint directory at the queue level enables checkpointing.

- ◆ All jobs submitted to the queue are checkpointable. LSF writes the checkpoint files, which contain job state information, to the checkpoint directory. The checkpoint directory can contain checkpoint files for multiple jobs.

NOTE: LSF does not delete the checkpoint files; you must perform file maintenance manually.

- ◆ If the queue administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every *chkpnt_period* during job execution.
- ◆ If a user specifies a checkpoint directory and checkpoint period at the job level with `bsub -k`, or modifies the job with `bmod`, the job-level values override the queue-level values.

The `brestart` command restarts checkpointed jobs that have stopped running.

Example

The following example shows a queue configured for periodic checkpointing in

```
lsb.queues:
```

```
Begin Queue
```

```
...
```

```
QUEUE_NAME=checkpoint
```

```
CHKPNT=mydir 240
```

```
DESCRIPTION=Automatically checkpoints jobs every 4 hours to mydir
```

```
...
```

```
End Queue
```

NOTE: The `bqueues` command displays the checkpoint period in seconds; the `lsb.queues` `CHKPNT` parameter defines the checkpoint period in minutes.

If the command `bchkpnt -k 123` is used to checkpoint and kill job 123, you can restart the job using the `brestart` command as shown in the following example:

```
brestart -q priority mydir 123
```

```
Job <456> is submitted to queue <priority>
```

LSF assigns a new job ID of 456, submits the job to the queue named "priority," and restarts the job.

Once job 456 is running, you can change the checkpoint period using the `bchkpnt` command:

```
bchkpnt -p 360 456
```

```
Job <456> is being checkpointed
```

NOTE: For a detailed description of the commands used with the job checkpoint and restart feature, see the *Platform LSF Configuration Reference*.

Checkpoint and restart executables

LSF controls checkpointing and restart by means of interfaces named `echkpnt` and `erestart`. By default, when a user specifies a checkpoint directory using `bsub -k` or `bmod -k` or submits a job to a queue that has a checkpoint directory specified, `echkpnt` sends checkpoint instructions to an executable named `echkpnt.default`.

For application-level job checkpoint and restart, you can write customized checkpoint and restart executables for each application that you use. The optional parameter `LSB_ECHKPNT_METHOD` specifies a checkpoint executable used for all jobs in the cluster. An LSF user can override this value when submitting a job.

NOTE: For a detailed description of how to write and configure application-level checkpoint and restart executables, see the *Platform LSF Configuration Reference*.

Job restart

LSF can restart a checkpointed job on a host other than the original execution host using the information saved in the checkpoint file to recreate the execution environment. Only jobs that have been checkpointed successfully can be restarted from a checkpoint file. When a job restarts, LSF performs the following actions:

- 1 LSF resubmits the job to its original queue as a new job and assigns a new job ID.
- 2 When a suitable host becomes available, LSF dispatches the job.
- 3 LSF recreates the execution environment from the checkpoint file.
- 4 LSF restarts the job from its last checkpoint. You can restart a job manually from the command line using `brestart`, automatically through configuration, or by migrating the job to a different host using `bmig`.

Requirements

To allow restart of a checkpointed job on a different host than the host on which the job originally ran, both the original and the new hosts must:

- ◆ Be binary compatible

- ◆ Run the same dot version of the operating system for predictable results
- ◆ Have network connectivity and read/execute permissions to the checkpoint and restart executables (in `LSF_SERVERDIR` by default)
- ◆ Have network connectivity and read/write permissions to the checkpoint directory and the checkpoint file
- ◆ Have access to all files open during job execution so that LSF can locate them using an absolute path name

Job migration

Job migration is the process of moving a checkpointable or rerunnable job from one host to another. This facilitates load balancing by moving jobs from a heavily-loaded host to a lightly-loaded host.

You can initiate job migration on demand (`bmig`) or automatically. To initiate job migration automatically, you can configure a migration threshold at the host or queue level.

NOTE: For a detailed description of the job migration feature and how to configure it, see the *Platform LSF Configuration Reference*.

Manual job migration

The `bmig` command migrates checkpointable or rerunnable jobs on demand. Jobs can be manually migrated by the job owner, queue administrator, and LSF administrator.

For example, to migrate a job with job ID 123:

```
bmig 123
Job <123> is being migrated
```

In this example, LSF migrates the job to the first available host.

Automatic job migration

Automatic job migration assumes that if a job is system-suspended (`SSUSP`) for an extended period of time, the execution host is probably heavily loaded. Configuring a queue-level or host-level migration threshold allows the job to progress and reduces the load on the host. You can use `bmig` at any time to override a configured migration threshold.

For example, at the queue level, in `lsb.queues`:

```
Begin Queue
...
MIG=30          # Migration threshold set to 30 mins
DESCRIPTION=Migrate suspended jobs after 30 mins
...
End Queue
```

At the host level, in `lsb.hosts`:

```
Begin Host
HOST_NAME      r1m   pg   MIG # Keywords
...
```

```
hostA      5.0  18  30
...
End Host
```

If you want to requeue migrated jobs instead of restarting or rerunning them, you can define the following parameters in `lsf.conf`:

- ◆ `LSB_MIG2PEND=1` requeues a job with the original submission time and priority
- ◆ `LSB_REQUEUE_TO_BOTTOM=1` requeues a job at the bottom of the queue, regardless of the submission time and priority

Job migration

Chunk Job Dispatch

Contents

- ◆ [About Job Chunking](#) on page 453
- ◆ [Configure Chunk Job Dispatch](#) on page 454
- ◆ [Submitting and Controlling Chunk Jobs](#) on page 456

About Job Chunking

LSF supports *job chunking*, where jobs with similar resource requirements submitted by the same user are grouped together for dispatch. The `CHUNK_JOB_SIZE` parameter in `lsb.queues` and `lsb.applications` specifies the maximum number of jobs allowed to be dispatched together in a *chunk job*.

Job chunking can have the following advantages:

- ◆ Reduces communication between `sbatchd` and `mbatchd`, and scheduling overhead in `mbatchd`
- ◆ Increases job throughput in `mbatchd` and more balanced CPU utilization on the execution hosts

All of the jobs in the chunk are dispatched as a unit rather than individually. Job execution is sequential, but each chunk job member is not necessarily executed in the order it was submitted.

Chunk job candidates

Jobs with the following characteristics are typical candidates for job chunking:

- ◆ Take between 1 and 2 minutes to run
- ◆ All require the same resource (for example a software license or a specific amount of memory)
- ◆ Do not specify a beginning time (`bsub -b`) or termination time (`bsub -t`)

Running jobs with these characteristics without chunking can underutilize resources because LSF spends more time scheduling and dispatching the jobs than actually running them.

Configuring a special high-priority queue for short jobs is not desirable because users may be tempted to send all of their jobs to this queue, knowing that it has high priority.

Configure Chunk Job Dispatch

CHUNK_JOB_SIZE (lsb.queues)

By default, `CHUNK_JOB_SIZE` is not enabled.

- 1 To configure a queue to dispatch chunk jobs, specify the `CHUNK_JOB_SIZE` parameter in the queue definition in `lsb.queues`.

For example, the following configures a queue named `chunk`, which dispatches up to 4 jobs in a chunk:

```
Begin Queue
QUEUE_NAME      = chunk
PRIORITY        = 50
CHUNK_JOB_SIZE  = 4
End Queue
```

Postrequisites: After adding `CHUNK_JOB_SIZE` to `lsb.queues`, use `badadmin reconfig` to reconfigure your cluster.

Chunk jobs and job throughput

Throughput can deteriorate if the chunk job size is too big. Performance may decrease on queues with `CHUNK_JOB_SIZE` greater than 30. You should evaluate the chunk job size on your own systems for best performance.

CHUNK_JOB_SIZE (lsb.applications)

By default, `CHUNK_JOB_SIZE` is not enabled. Enabling application-level job chunking overrides queue-level job chunking.

- 1 To configure an application profile to chunk jobs together, specify the `CHUNK_JOB_SIZE` parameter in the application profile definition in `lsb.applications`.
Specify `CHUNK_JOB_SIZE=1` to disable job chunking for the application. This value overrides chunk job dispatch configured in the queue.

Postrequisites: After adding `CHUNK_JOB_SIZE` to `lsb.applications`, use `badadmin reconfig` to reconfigure your cluster.

CHUNK_JOB_DURATION (lsb.params)

If `CHUNK_JOB_DURATION` is defined in the file `lsb.params`, a job submitted to a chunk job queue will be chunked under the following conditions:

- ◆ A job-level CPU limit or run time limit is specified (`bsub -c` or `-W`), or
- ◆ An application-level CPU limit, run time limit, or run time estimate is specified (`CPULIMIT`, `RUNLIMIT`, or `RUNTIME` in `lsb.applications`), or

- ◆ A queue-level CPU limit or run time limit is specified (CPULIMIT or RUNLIMIT in `lsb.queues`),

and the values of the CPU limit, run time limit, and run time estimate are all less than or equal to the `CHUNK_JOB_DURATION`.

Jobs are not chunked if:

- ◆ The CPU limit, run time limit, or run time estimate is greater than the value of `CHUNK_JOB_DURATION`, or
- ◆ No CPU limit, no run time limit, and no run time estimate are specified.

The value of `CHUNK_JOB_DURATION` is displayed by `bparams -l`.

-
- 1 After adding `CHUNK_JOB_DURATION` to `lsb.params`, use `badmin reconfig` to reconfigure your cluster.

By default, `CHUNK_JOB_DURATION` is not enabled.

Restrictions on chunk jobs

`CHUNK_JOB_SIZE` is ignored and jobs are not chunked under the following conditions:

- ◆ Interactive queues (`INTERACTIVE = ONLY` parameter)
- ◆ CPU limit greater than 30 minutes (CPULIMIT parameter in `lsb.queues` or `lsb.applications`). If `CHUNK_JOB_DURATION` is set in `lsb.params`, the job is chunked only if it is submitted with a CPU limit that is less than or equal to the value of `CHUNK_JOB_DURATION` (`bsub -c`)
- ◆ Run limit greater than 30 minutes (RUNLIMIT parameter in `lsb.queues` or `lsb.applications`). If `CHUNK_JOB_DURATION` is set in `lsb.params`, the job is chunked only if it is submitted with a run limit that is less than or equal to the value of `CHUNK_JOB_DURATION` (`bsub -W`)
- ◆ Run time estimate greater than 30 minutes (RUNTIME parameter in `lsb.applications`)

Jobs submitted with the following `bsub` options are not chunked; they are dispatched individually:

- ◆ `-I` (interactive jobs)
- ◆ `-c` (jobs with CPU limit greater than 30)
- ◆ `-W` (jobs with run limit greater than 30 minutes)
- ◆ `-app` (jobs associated with an application profile that specifies a run time estimate or run time limit greater than 30 minutes, or a CPU limit greater than 30). `CHUNK_JOB_SIZE` is either not specified in the application, or `CHUNK_JOB_SIZE=1`, which disables chunk job dispatch configured in the queue.

Submitting and Controlling Chunk Jobs

When a job is submitted to a queue or application profile configured with the `CHUNK_JOB_SIZE` parameter, LSF attempts to place the job in an existing chunk. A job is added to an existing chunk if it has the same characteristics as the first job in the chunk:

- ◆ Submitting user
- ◆ Resource requirements
- ◆ Host requirements
- ◆ Queue or application profile
- ◆ Job priority

If a suitable host is found to run the job, but there is no chunk available with the same characteristics, LSF creates a new chunk.

Resources reserved for any member of the chunk are reserved at the time the chunk is dispatched and held until the whole chunk finishes running. Other jobs requiring the same resources are not dispatched until the chunk job is done.

For example, if all jobs in the chunk require a software license, the license is checked out and each chunk job member uses it in turn. The license is not released until the last chunk job member is finished running.

WAIT status

When `sbatchd` receives a chunk job, it will not start all member jobs at once. A chunk job occupies a single job slot. Even if other slots are available, the chunk job members must run one at a time in the job slot they occupy. The remaining jobs in the chunk that are waiting to run are displayed as `WAIT` by `bjobs`. Any jobs in `WAIT` status are included in the count of pending jobs by `bqueues` and `busers`. The `bhosts` command shows the single job slot occupied by the entire chunk job in the number of jobs shown in the `NJOBS` column.

The `bhist -l` command shows jobs in `WAIT` status as `Waiting ...`

The `bjobs -l` command does not display a `WAIT` reason in the list of pending jobs.

Controlling chunk jobs

Job controls affect the state of the members of a chunk job. You can perform the following actions on jobs in a chunk job:

Action (Command)	Job State	Effect on Job (State)
Suspend (<code>bstop</code>)	PEND	Removed from chunk (PSUSP)
	RUN	All jobs in the chunk are suspended (NRUN -1, NSUSP +1)
	USUSP	No change
	WAIT	Removed from chunk (PSUSP)
Kill (<code>bkill</code>)	PEND	Removed from chunk (NJOBS -1, PEND -1)
	RUN	Job finishes, next job in the chunk starts if one exists (NJOBS -1, PEND -1)
	USUSP	Job finishes, next job in the chunk starts if one exists (NJOBS -1, PEND -1, SUSP -1, RUN +1)

Action (Command)	Job State	Effect on Job (State)
	WAIT	Job finishes (NJOBS-1, PEND -1)
Resume (bresume)	USUSP	Entire chunk is resumed (RUN +1, USUSP -1)
Migrate (bmig)	WAIT	Removed from chunk
Switch queue (bswitch)	RUN	Job is removed from the chunk and switched; all other WAIT jobs are requeued to PEND
	WAIT	Only the WAIT job is removed from the chunk and switched, and requeued to PEND
Checkpoint (bchkpnt)	RUN	Job is checkpointed normally
Modify (bmod)	PEND	Removed from the chunk to be scheduled later

Migrating jobs with `bmig` will change the dispatch sequence of the chunk job members. They will not be redispached in the order they were originally submitted.

Rerunnable chunk jobs

If the execution host becomes unavailable, rerunnable chunk job members are removed from the queue and dispatched to a different execution host.

See Chapter 27, “[Job Requeue and Job Rerun](#)” for more information about rerunnable jobs.

Checkpointing chunk jobs

Only running chunk jobs can be checkpointed. If `bchkpnt -k` is used, the job is also killed after the checkpoint file has been created. If chunk job in WAIT state is checkpointed, `mbatchd` rejects the checkpoint request.

See Chapter 28, “[Job Checkpoint, Restart, and Migration](#)” for more information about checkpointing jobs.

Fairshare policies and chunk jobs

Fairshare queues can use job chunking. Jobs are accumulated in the chunk job so that priority is assigned to jobs correctly according to the fairshare policy that applies to each user. Jobs belonging to other users are dispatched in other chunks.

TERMINATE_WHEN job control action

If the `TERMINATE_WHEN` job control action is applied to a chunk job, `sbatchd` kills the chunk job element that is running and puts the rest of the waiting elements into pending state to be rescheduled later.

Enforce resource usage limits on chunk jobs

By default, resource usage limits are not enforced for chunk jobs because chunk jobs are typically too short to allow LSF to collect resource usage.

- 1 To enforce resource limits for chunk jobs, define `LSB_CHUNK_RUSAGE=Y` in `lsf.conf`. Limits may not be enforced for chunk jobs that take less than a minute to run.

Submitting and Controlling Chunk Jobs

Job Arrays

LSF provides a structure called a job array that allows a sequence of jobs that share the same executable and resource requirements, but have different input files, to be submitted, controlled, and monitored as a single unit. Using the standard LSF commands, you can also control and monitor individual jobs and groups of jobs submitted from a job array.

After the job array is submitted, LSF independently schedules and dispatches the individual jobs. Each job submitted from a job array shares the same job ID as the job array and are uniquely referenced using an array index. The dimension and structure of a job array is defined when the job array is created.

Contents

- ◆ [Create a Job Array](#) on page 459
- ◆ [Handling Input and Output Files](#) on page 461
- ◆ [Job Array Dependencies](#) on page 463
- ◆ [Monitoring Job Arrays](#) on page 463
- ◆ [Controlling Job Arrays](#) on page 465
- ◆ [Requeuing a Job Array](#) on page 466
- ◆ [Job Array Job Slot Limit](#) on page 467

Create a Job Array

A job array is created at job submission time using the `-J` option of `bsub`.

- 1 For example, the following command creates a job array named `myArray` made up of 1000 jobs.

```
bsub -J "myArray[1-1000]" myJob
Job <123> is submitted to default queue <normal>.
```

Syntax

The `bsub` syntax used to create a job array follows:

```
bsub -J "arrayName[indexList, ...]" myJob
```

Where:

-J "arrayName[indexList, ...]"

Names and creates the job array. The square brackets, [], around `indexList` must be entered exactly as shown and the job array name specification must be enclosed in quotes. Commas (,) are used to separate multiple `indexList` entries. The maximum length of this specification is 255 characters.

arrayName

User specified string used to identify the job array. Valid values are any combination of the following characters:

```
a-z | A-Z | 0-9 | . | - | _
```

indexList = start[-end[:step]]

Specifies the size and dimension of the job array, where:

start: Specifies the start of a range of indices. Can also be used to specify an individual index. Valid values are unique positive integers. For example, [1-5] and [1, 2, 3, 4, 5] specify 5 jobs with indices 1 through 5.

end: Specifies the end of a range of indices. Valid values are unique positive integers.

step: Specifies the value to increment the indices in a range. Indices begin at `start`, increment by the value of `step`, and do not increment past the value of `end`. The default value is 1. Valid values are positive integers. For example, [1-10:2] specifies a range of 1-10 with step value 2 creating indices 1, 3, 5, 7, and 9.

After the job array is created (submitted), individual jobs are referenced using the job array name or job ID and an index value. For example, both of the following series of job array statements refer to jobs submitted from a job array named `myArray` which is made up of 1000 jobs and has a job ID of 123:

```
myArray[1], myArray[2], myArray[3], ..., myArray[1000]  
123[1], 123[2], 123[3], ..., 123[1000]
```

Change the maximum size of a job array

A large job array allows a user to submit a large number of jobs to the system with a single job submission.

By default, the maximum number of jobs in a job array is 1000, which means the maximum size of a job array can never exceed 1000 jobs.

-
- 1 To make a change to the maximum job array value, set `MAX_JOB_ARRAY_SIZE` in `lsb.params` to any positive integer between 1 and 2147483646. The maximum number of jobs in a job array cannot exceed the value set by `MAX_JOB_ARRAY_SIZE`.
-

Handling Input and Output Files

LSF provides methods for coordinating individual input and output files for the multiple jobs created when submitting a job array. These methods require your input files to be prepared uniformly. To accommodate an executable that uses standard input and standard output, LSF provides runtime variables (`%I` and `%J`) that are expanded at runtime. To accommodate an executable that reads command line arguments, LSF provides an environment variable (`LSB_JOBINDEX`) that is set in the execution environment.

Methods

- ◆ [Redirecting Standard Input and Output](#) on page 461
- ◆ [Passing Arguments on the Command Line](#) on page 462

Prepare input files

LSF needs all the input files for the jobs in your job array to be located in the same directory. By default LSF assumes the current working directory (CWD); the directory from where `bsub` was issued.

-
- 1 To override CWD, specify an absolute path when submitting the job array. Each file name consists of two parts, a consistent name string and a variable integer that corresponds directly to an array index. For example, the following file names are valid input file names for a job array. They are made up of the consistent name `input` and integers that correspond to job array indices from 1 to 1000:
`input.1, input.2, input.3, ..., input.1000`
-

Redirecting Standard Input and Output

The variables `%I` and `%J` are used as substitution strings to support file redirection for jobs submitted from a job array. At execution time, `%I` is expanded to provide the job array index value of the current job, and `%J` is expanded to provide the job ID of the job array.

Redirect standard input

- 1 Use the `-i` option of `bsub` and the `%I` variable when your executable reads from standard input.

To use `%I`, all the input files must be named consistently with a variable part that corresponds to the indices of the job array. For example:

```
input.1, input.2, input.3, ..., input.N
```

For example, the following command submits a job array of 1000 jobs whose input files are named `input.1`, `input.2`, `input.3`, ..., `input.1000` and located in the current working directory:

```
bsub -J "myArray[1-1000]" -i "input.%I" myJob
```

Redirect standard output and error

- 1 Use the `-o` option of `bsub` and the `%I` and `%J` variables when your executable writes to standard output and error.

- a To create an output file that corresponds to each job submitted from a job array, specify `%I` as part of the output file name.

For example, the following command submits a job array of 1000 jobs whose output files will be located in CWD and named `output.1`, `output.2`, `output.3`, ..., `output.1000`:

```
bsub -J "myArray[1-1000]" -o "output.%I" myJob
```

- b To create output files that include the job array job ID as part of the file name specify `%J`.

For example, the following command submits a job array of 1000 jobs whose output files will be located in CWD and named `output.123.1`, `output.123.2`, `output.123.3`, ..., `output.123.1000`. The job ID of the job array is 123.

```
bsub -J "myArray[1-1000]" -o "output.%J.%I" myJob
```

Passing Arguments on the Command Line

The environment variable `LSB_JOBINDEX` is used as a substitution string to support passing job array indices on the command line. When the job is dispatched, LSF sets `LSB_JOBINDEX` in the execution environment to the job array index of the current job. `LSB_JOBINDEX` is set for all jobs. For non-array jobs, `LSB_JOBINDEX` is set to zero (0).

To use `LSB_JOBINDEX`, all the input files must be named consistently and with a variable part that corresponds to the indices of the job array. For example:

```
input.1, input.2, input.3, ..., input.N
```

You must escape `LSB_JOBINDEX` with a backslash, `\`, to prevent the shell interpreting `bsub` from expanding the variable. For example, the following command submits a job array of 1000 jobs whose input files are named `input.1`,

input.2, input.3, ..., input.1000 and located in the current working directory. The executable is being passed an argument that specifies the name of the input files:

```
bsub -J "myArray[1-1000]" myJob -f input.\$LSB_JOBINDEX
```

Job Array Dependencies

Like all jobs in LSF, a job array can be dependent on the completion or partial completion of a job or another job array. A number of job-array-specific dependency conditions are provided by LSF.

Set a whole array dependency

- 1 To make a job array dependent on the completion of a job or another job array use the `-w "dependency_condition"` option of `bsub`.

For example, to have an array dependent on the completion of a job or job array with job ID 123, use the following command:

```
bsub -w "done(123)" -J "myArray2[1-1000]" myJob
```

Set a partial array dependency

- 1 To make a job or job array dependent on an existing job array, use one of the following dependency conditions.

Condition	Description
<code>numrun(jobArrayJobId, op num)</code>	Evaluate the number of jobs in RUN state
<code>numpend(jobArrayJobId, op num)</code>	Evaluate the number of jobs in PEND state
<code>numdone(jobArrayJobId, op num)</code>	Evaluate the number of jobs in DONE state
<code>numexit(jobArrayJobId, op num)</code>	Evaluate the number of jobs in EXIT state
<code>numended(jobArrayJobId, op num)</code>	Evaluate the number of jobs in DONE and EXIT state
<code>numhold(jobArrayJobId, op num)</code>	Evaluate the number of jobs in PSUSP state
<code>numstart(jobArrayJobId, op num)</code>	Evaluate the number of jobs in RUN and SSUSP and USUSP state

- 2 Use one the following operators (`op`) combined with a positive integer (`num`) to build a condition:

```
== | > | < | >= | <= | !=
```

Optionally, an asterisk (*) can be used in place of `num` to mean all jobs submitted from the job array.

For example, to start a job named `myJob` when 100 or more elements in a job array with job ID 123 have completed successfully:

```
bsub -w "numdone(123, >= 100)" myJob
```

Monitoring Job Arrays

Use `bjobs` and `bhist` to monitor the current and past status of job arrays.

Individual job status

Display job array status

- 1 To display summary information about the currently running jobs submitted from a job array, use the `-A` option of `bjobs`.

For example, a job array of 10 jobs with job ID 123:

```
bjobs -A 123
JOBID  ARRAY_SPEC  OWNER  NJOBS  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
123    myArra[1-10]  user1   10     3     3     3   4     0     0     0     0
```

Individual job status

Display current job status

- 1 To display the status of the individual jobs submitted from a job array, specify the job array job ID with `bjobs`. For jobs submitted from a job array, `JOBID` displays the job array job ID, and `JOBNAME` displays the job array name and the index value of each job.

For example, to view a job array with job ID 123:

```
bjobs 123
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
123    user1  DONE  default  hostA      hostC      myArray[1]  Feb 29 12:34
123    user1  DONE  default  hostA      hostQ      myArray[2]  Feb 29 12:34
123    user1  DONE  default  hostA      hostB      myArray[3]  Feb 29 12:34
123    user1  RUN   default  hostA      hostC      myArray[4]  Feb 29 12:34
123    user1  RUN   default  hostA      hostL      myArray[5]  Feb 29 12:34
123    user1  RUN   default  hostA      hostB      myArray[6]  Feb 29 12:34
123    user1  RUN   default  hostA      hostQ      myArray[7]  Feb 29 12:34
123    user1  PEND  default  hostA      hostQ      myArray[8]  Feb 29 12:34
123    user1  PEND  default  hostA      hostQ      myArray[9]  Feb 29 12:34
123    user1  PEND  default  hostA      hostQ      myArray[10] Feb 29 12:34
```

Display past job status

- 1 To display the past status of the individual jobs submitted from a job array, specify the job array job ID with `bhist`.

For example, to view the history of a job array with job ID 456:

```
bhist 456
Summary of time in seconds spent in various states:
JOBID  USER  JOB_NAME  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
456[1] user1  *rray[1]  14    0      65    0      0      0      79
456[2] user1  *rray[2]  74    0      25    0      0      0      99
456[3] user1  *rray[3]  121   0      26    0      0      0     147
456[4] user1  *rray[4]  167   0      30    0      0      0     197
456[5] user1  *rray[5]  214   0      29    0      0      0     243
456[6] user1  *rray[6]  250   0      35    0      0      0     285
456[7] user1  *rray[7]  295   0      33    0      0      0     328
```


456[8]	user1	*rray[8]	339	0	29	0	0	0	368
456[9]	user1	*rray[9]	356	0	26	0	0	0	382
456[10]	user1	*rray[10]	375	0	24	0	0	0	399

Specific job status

Display the current status of a specific job

- 1 To display the current status of a specific job submitted from a job array, specify in quotes, the job array job ID and an index value with `bjobs`.

For example, the status of the 5th job in a job array with job ID 123:

```
bjobs "123[5]"
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
123    user1  RUN   default hostA      hostL      myArray[5] Feb 29 12:34
```

Display the past status of a specific job

- 1 To display the past status of a specific job submitted from a job array, specify, in quotes, the job array job ID and an index value with `bhist`.

For example, the status of the 5th job in a job array with job ID 456:

```
bhist "456[5]"
Summary of time in seconds spent in various states:
JOBID  USER  JOB_NAME  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
456[5] user1  *rray[5]  214   0      29   0      0      0      243
```

Controlling Job Arrays

You can control the whole array, all the jobs submitted from the job array, with a single command. LSF also provides the ability to control individual jobs and groups of jobs submitted from a job array. When issuing commands against a job array, use the job array job ID instead of the job array name. Job names are not unique in LSF, and issuing a command using a job array name may result in unpredictable behavior.

Most LSF commands allow operation on both the whole job array, individual jobs, and groups of jobs. These commands include `bkill`, `bstop`, `bresume`, and `bmod`.

Some commands only allow operation on individual jobs submitted from a job array. These commands include `btop`, `bbot`, and `bswitch`.

Control a whole array

- 1 To control the whole job array, specify the command as you would for a single job using only the job ID.

For example, to kill a job array with job ID 123:

```
bkill 123
```

Control individual jobs

- 1 To control an individual job submitted from a job array, specify the command using the job ID of the job array and the index value of the corresponding job. The job ID and index value must be enclosed in quotes.

For example, to kill the 5th job in a job array with job ID 123:

```
bkill "123[5]"
```

Control groups of jobs

- 1 To control a group of jobs submitted from a job array, specify the command as you would for an individual job and use `indexList` syntax to indicate the jobs.

For example, to kill jobs 1-5, 239, and 487 in a job array with job ID 123:

```
bkill "123[1-5, 239, 487]"
```

Requeuing a Job Array

Use `brequeue` to requeue a job array. When the job is requeued, it is assigned the `PEND` status and the job's new position in the queue is after other jobs of the same priority. You can requeue:

- ◆ Jobs in `DONE` job state
- ◆ Jobs in `EXIT` job state
- ◆ All jobs regardless of job state in a job array.
- ◆ `EXIT`, `RUN`, `DONE` jobs to `PSUSP` state
- ◆ Jobs in `RUN` job state

`brequeue` is not supported across clusters.

Requeue jobs in `DONE` state

- 1 To requeue `DONE` jobs use the `-d` option of `brequeue`.

For example, the command `brequeue -J "myarray[1-10]" -d 123` requeues jobs with job ID 123 and `DONE` status.

Requeue Jobs in EXIT state

- 1 To requeue EXIT jobs use the `-e` option of `brequeue`.
For example, the command `brequeue -J "myarray[1-10]" -e 123` requeues jobs with job ID 123 and EXIT status.

Requeue all jobs in an array regardless of job state

- 1 A submitted job array can have jobs that have different job states. To requeue all the jobs in an array regardless of any job's state, use the `-a` option of `brequeue`.
For example, the command `brequeue -J "myarray[1-10]" -a 123` requeues all jobs in a job array with job ID 123 regardless of their job state.

Requeue RUN jobs to PSUSP state

- 1 To requeue RUN jobs to PSUSP state, use the `-H` option of `brequeue`.
For example, the command `brequeue -J "myarray[1-10]" -H 123` requeues to PSUSP RUN status jobs with job ID 123.

Requeue jobs in RUN state

- 1 To requeue RUN jobs use the `-r` option of `brequeue`.
For example, the command `brequeue -J "myarray[1-10]" -r 123` requeues jobs with job ID 123 and RUN status.

Job Array Job Slot Limit

The job array job slot limit is used to specify the maximum number of jobs submitted from a job array that are allowed to run at any one time. A job array allows a large number of jobs to be submitted with one command, potentially flooding a system, and job slot limits provide a way to limit the impact a job array may have on a system. Job array job slot limits are specified using the following syntax:

```
bsub -J "job_array_name[index_list]%job_slot_limit" myJob
```

where:

- %job_slot_limit** Specifies the maximum number of jobs allowed to run at any one time. The percent sign (%) must be entered exactly as shown. Valid values are positive integers less than the maximum index value of the job array.

Setting a job array job slot limit

Set a job array slot limit at submission

- 1 Use the `bsub` command to set a job slot limit at the time of submission.
To set a job array job slot limit of 100 jobs for a job array of 1000 jobs:

```
bsub -J "job_array_name[1000]%100" myJob
```

Set a job array slot limit after submission

- 1 Use the `bmod` command to set a job slot limit after submission.
For example, to set a job array job slot limit of 100 jobs for an array with job ID 123:

```
bmod -J "%100" 123
```

Change a job array job slot limit

Changing a job array job slot limit is the same as setting it after submission.

- 1 Use the `bmod` command to change a job slot limit after submission.
For example, to change a job array job slot limit to 250 for a job array with job ID 123:

```
bmod -J "%250" 123
```

View a job array job slot limit

- 1 To view job array job slot limits use the `-A` and `-l` options of `bjobs`. The job array job slot limit is displayed in the Job Name field in the same format in which it was set.
For example, the following output displays the job array job slot limit of 100 for a job array with job ID 123:

```
bjobs -A -l 123
Job <123>, Job Name <myArray[1-1000]%100>, User <user1>, Project <default>, Sta
tus <PEND>, Queue <normal>, Job Priority <20>, Command <my
Job>
Wed Feb 29 12:34:56: Submitted from host <hostA>, CWD <${HOME}>;

COUNTERS:
NJOB  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
10     9     0     1     0     0     0     0
```

Running Parallel Jobs

Contents

- ◆ [How LSF Runs Parallel Jobs](#) on page 469
- ◆ [Preparing Your Environment to Submit Parallel Jobs to LSF](#) on page 470
- ◆ [Submitting Parallel Jobs](#) on page 470
- ◆ [Starting Parallel Tasks with LSF Utilities](#) on page 471
- ◆ [Job Slot Limits For Parallel Jobs](#) on page 472
- ◆ [Specifying a Minimum and Maximum Number of Processors](#) on page 472
- ◆ [Specifying a First Execution Host](#) on page 473
- ◆ [Controlling Processor Allocation Across Hosts](#) on page 474
- ◆ [Running Parallel Processes on Homogeneous Hosts](#) on page 477
- ◆ [Limiting the Number of Processors Allocated](#) on page 478
- ◆ [Reserving Processors](#) on page 481
- ◆ [Reserving Memory for Pending Parallel Jobs](#) on page 482
- ◆ [Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots](#) on page 483
- ◆ [Parallel Fairshare](#) on page 492
- ◆ [How Deadline Constraint Scheduling Works For Parallel Jobs](#) on page 493
- ◆ [Optimized Preemption of Parallel Jobs](#) on page 493

How LSF Runs Parallel Jobs

When LSF runs a job, the `LSB_HOSTS` variable is set to the names of the hosts running the batch job. For a parallel batch job, `LSB_HOSTS` contains the complete list of hosts that LSF has allocated to that job.

LSF starts one controlling process for the parallel batch job on the first host in the host list. It is up to your parallel application to read the `LSB_HOSTS` environment variable to get the list of hosts, and start the parallel job components on all the other allocated hosts.

LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs.

Preparing Your Environment to Submit Parallel Jobs to LSF

Getting the host list

Some applications can take this list of hosts directly as a command line parameter. For other applications, you may need to process the host list.

Example

The following example shows a `/bin/sh` script that processes all the hosts in the host list, including identifying the host where the job script is executing.

```
#!/bin/sh
# Process the list of host names in LSB_HOSTS

for host in $LSB_HOSTS ; do
handle_host $host
done
```

Parallel job scripts

Each parallel programming package has different requirements for specifying and communicating with all the hosts used by a parallel job. LSF is not tailored to work with a specific parallel programming package. Instead, LSF provides a generic interface so that any parallel package can be supported by writing shell scripts or wrapper programs.

You can modify these scripts to support more parallel packages.

For more information, see [Submitting Parallel Jobs](#) on page 470

Use a job starter

You can configure the script into your queue as a job starter, and then all users can submit parallel jobs without having to type the script name. See [Queue-Level Job Starters](#) on page 540 for more information about job starters.

1 To see if your queue already has a job starter defined, run `bqueues -l`.

Submitting Parallel Jobs

LSF can allocate more than one host or processor to run a job and automatically keeps track of the job status, while a parallel job is running.

Specify the number of processors

When submitting a parallel job that requires multiple processors, you can specify the exact number of processors to use.

1 To submit a parallel job, use `bsub -n` and specify the number of processors the job requires.

2 To submit jobs based on the number of available job slots instead of the number of processors, use `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`.

For example:

```
bsub -n 4 myjob
```

submits myjob as a parallel job. The job is started when 4 job slots are available.

Starting Parallel Tasks with LSF Utilities

For simple parallel jobs you can use LSF utilities to start parts of the job on other hosts. Because LSF utilities handle signals transparently, LSF can suspend and resume all components of your job without additional programming.

Running parallel tasks with lsgrun

The simplest parallel job runs an identical copy of the executable on every host. The `lsgrun` command takes a list of host names and runs the specified task on each host. The `lsgrun -p` command specifies that the task should be run in parallel on each host.

Example

This example submits a job that uses `lsgrun` to run `myjob` on all the selected hosts in parallel:

```
bsub -n 10 'lsgrun -p -m "$LSB_HOSTS" myjob'
Job <3856> is submitted to default queue <normal>.
```

For more complicated jobs, you can write a shell script that runs `lsgun` in the background to start each component.

Running parallel tasks with the blaunch distributed application framework

Most MPI implementations and many distributed applications use `rsh` and `ssh` as their task launching mechanism. The `blaunch` command provides a drop-in replacement for `rsh` and `ssh` as a transparent method for launching parallel and distributed applications within LSF.

Similar to the `lsgun` command, `blaunch` transparently connects directly to the RES/SBD on the remote host, and subsequently creates and tracks the remote tasks, and provides the connection back to LSF. There is no need to insert `pam` or `taskstarter` into the `rsh` or `ssh` calling sequence, or configure any wrapper scripts.

IMPORTANT: You cannot run `blaunch` directly from the command line.

`blaunch` only works within an LSF job; it can only be used to launch tasks on remote hosts that are part of a job allocation. It cannot be used as a standalone command. On success `blaunch` exits with 0.

`blaunch` is not supported on Windows.

See *Using Platform LSF HPC* for more information about using the `blaunch` distributed application framework.

Submitting jobs with blaunch

Use `bsub` to call `blaunch`, or to invoke a job script that calls `blaunch`. The `blaunch` command assumes that `bsub -n` implies one remote task per job slot.

- ◆ Submit a parallel job:


```
bsub -n 4 blaunch myjob
```
- ◆ Submit a parallel job to launch tasks on a specific host:


```
bsub -n 4 blaunch hostA myjob
```

- ◆ Submit a job with a host list:
`bsub -n 4 blaunch -z "hostA hostB" myjob`
- ◆ Submit a job with a host file:
`bsub -n 4 blaunch -u ./hostfile myjob`
- ◆ Submit a job to an application profile
`bsub -n 4 -app pjob blaunch myjob`

Job Slot Limits For Parallel Jobs

A job slot is the basic unit of processor allocation in LSF. A sequential job uses one job slot. A parallel job that has N components (tasks) uses N job slots, which can span multiple hosts.

By default, running and suspended jobs count against the job slot limits for queues, users, hosts, and processors that they are associated with.

With processor reservation, job slots reserved by pending jobs also count against all job slot limits.

When backfilling occurs, the job slots used by backfill jobs count against the job slot limits for the queues and users, but not hosts or processors. This means when a pending job and a running job occupy the same physical job slot on a host, both jobs count towards the queue limit, but only the pending job counts towards host limit.

Specifying a Minimum and Maximum Number of Processors

When submitting a parallel job, you can also specify a minimum number and a maximum number of processors.

If you specify a maximum and minimum number of processors, the job will start as soon as the minimum number of processors is available, but it will use up to the maximum number of processors, depending on how many processors are available at the time. Once the job starts running, no more processors will be allocated to it even though more may be available later on.

Jobs that request fewer processors than the minimum PROCLIMIT defined for the queue or application profile to which the job is submitted, or more processors than the maximum PROCLIMIT are rejected. If the job requests minimum and maximum processors, the maximum requested cannot be less than the minimum PROCLIMIT, and the minimum requested cannot be more than the maximum PROCLIMIT.

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the job specifies a maximum and minimum number of job slots instead of processors.

Syntax

```
bsub -n min_proc[,max_proc]
```

Example

```
bsub -n 4,16 myjob
```


At most, 16 processors can be allocated to this job. If there are less than 16 processors eligible to run the job, this job can still be started as long as the number of eligible processors is greater than or equal to 4.

Specifying a First Execution Host

In general, the first execution host satisfies certain resource requirements that might not be present on other available hosts.

By default, LSF selects the first execution host dynamically according to the resource availability and host load for a parallel job. Alternatively, you can specify one or more first execution host candidates so that LSF selects one of the candidates as the first execution host.

Specify a first execution host

To specify one or more hosts or host groups as first execution host candidates, add the (!) symbol after the host name. You can specify first execution host candidates at job submission, or in the queue definition.

Job level

- 1 Use the `-m` option of `bsub`:

```
bsub -n 32 -m "hostA! hostB hostgroup1! hostC" myjob
```

The scheduler will select either `hostA` or a host defined in `hostgroup1` as the first execution host, based on the job's resource requirements and host availability.

- 2 In a MultiCluster environment, insert the (!) symbol after the cluster name, as shown in the following example:

```
bsub -n 2 -m "host2@cluster2! host3@cluster2" my_parallel_job
```

Queue level

The queue-level specification of first execution host candidates applies to all jobs submitted to the queue.

- 1 Specify the first execution host candidates in the list of hosts in the `HOSTS` parameter in `lsb.queues`:

```
HOSTS = hostA! hostB hostgroup1! hostC
```

Rules

Follow these guidelines when you specify first execution host candidates:

- ◆ If you specify a host group, you must first define the host group in the file `lsb.hosts`.
- ◆ Do not specify a dynamic host group as a first execution host.
- ◆ Do not specify "all," "allremote," or "others," or a host partition as a first execution host.
- ◆ Do not specify a preference (+) for a host identified by (!) as a first execution host candidate.
- ◆ For each parallel job, specify enough regular hosts to satisfy the CPU requirement for the job. Once LSF selects a first execution host for the current job, the other first execution host candidates

Controlling Processor Allocation Across Hosts

- ❖ Become unavailable to the current job
- ❖ Remain available to other jobs as either regular or first execution hosts
- ◆ You cannot specify first execution host candidates when you use the `brun` command.

If the first execution host is incorrect at job submission, the job will be rejected. If incorrect configurations exist at the queue level, warning messages will be logged and

ed when LSF starts, restarts, or is reconfigured.

Job chunking

Specifying first execution host candidates affects job chunking. For example, the following jobs have different job requirements, and will not be placed in the same job chunk:

```
bsub -n 2 -m "hostA! hostB hostC" myjob
bsub -n 2 -m "hostA hostB hostC" myjob
bsub -n 2 -m "hostA hostB! hostC" myjob
```

The requirements of each job in this example are:

- ◆ Job 1 must start on hostA
- ◆ Job 2 can start and run on hostA, hostB, or hostC
- ◆ Job 3 must start on hostB

For job chunking, all jobs must request the same hosts *and* the same first execution hosts (if specified). Jobs that specify a host preference must all specify the same preference.

Resource reservation

If you specify first execution host candidates at the job or queue level, LSF will try to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it will not reserve slots on any other hosts.

Controlling Processor Allocation Across Hosts

Sometimes you need to control how the selected processors for a parallel job are distributed across the hosts in the cluster.

You can control this at the job level or at the queue level. The queue specification is ignored if your job specifies its own locality.

Specifying parallel job locality at the job level

By default, LSF will allocate the required processors for the job from the available set of processors.

A parallel job may span multiple hosts, with a specifiable number of processes allocated to each host. A job may be scheduled on to a single multiprocessor host to take advantage of its efficient shared memory, or spread out on to multiple hosts to take advantage of their aggregate memory and swap space. Flexible spanning may also be used to achieve parallel I/O.

You are able to specify “select all the processors for this parallel batch job on the same host”, or “do not choose more than *n* processors on one host” by using the `span` section in the resource requirement string (`bsub -R` or `RES_REQ` in the queue definition in `lsb.queues`).

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the `span` string is used to control the number of job slots instead of processors.

Syntax

The `span` string supports the following syntax:

- span[hosts=1]** Indicates that all the processors allocated to this job must be on the same host.
- span[ptile=value]** Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:

- ◆ Default `ptile` value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:

```
span[ptile=4]
```

- ◆ Predefined `ptile` value, specified by '!'. The following example uses the predefined maximum job slot limit `lsb.hosts (MXJ)` per host type/model as its value:

```
span[ptile='!']
```

TIP: If the host or host type/model does not define `MXJ`, the default predefined `ptile` value is 1.

- ◆ Predefined `ptile` value with optional multiple `ptile` values, per host type or host model:

- ❖ For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type `HP` or `SGI`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host types:

```
span[ptile='!',HP:8,SGI:8,LINUX:2] same[type]
```

- ❖ For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host models:

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

- span[hosts=-1]** Disables `span` setting in the queue. LSF allocates the required processors for the job from the available set of processors.

Specifying multiple `ptile` values

In a `span` string with multiple `ptile` values, you must specify a predefined default value (`ptile='!'`) and either host model or host type.

You can specify both type and model in the `same` section in the resource requirement string, but the `ptile` values must be the same type.

If you specify `same[type:model]`, you *cannot* specify a predefined `ptile` value (!) in the `span` section.

RESTRICTION: Under `bash 3.0`, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined `ptile` value (`ptile='!'`), use the `+H` option to disable '!' style history substitution in `bash (sh +H)`.

Controlling Processor Allocation Across Hosts

The following `span` strings are valid:

```
same[type:model] span[ptile=LINUX:2,SGI:4]
```

LINUX and SGI are both host types and can appear in the same `span` string.

```
same[type:model] span[ptile=PC233:2,PC1133:4]
```

PC233 and PC1133 are both host models and can appear in the same `span` string.

You cannot mix host model and host type in the same `span` string. The following `span` strings are *not* correct:

```
span[ptile='!',LINUX:2,PC1133:4] same[model]
```

```
span[ptile='!',LINUX:2,PC1133:4] same[type]
```

The LINUX host type and PC1133 host model cannot appear in the same `span` string.

Multiple `ptile` values for a host type

For host type, you must specify `same[type]` in the resource requirement. For example:

```
span[ptile='!',HP:8,SGI:8,LINUX:2] same[type]
```

The job requests 8 processors on a host of type HP or SGI, and 2 processors on a host of type LINUX, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host types.

Multiple `ptile` values for a host model

For host model, you must specify `same[model]` in the resource requirement. For example:

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

The job requests 4 processors on hosts of model PC1133, and 2 processors on hosts of model PC233, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host models.

Examples

```
bsub -n 4 -R "span[hosts=1]" myjob
```

Runs the job on a host that has at least 4 processors currently eligible to run the 4 components of this job.

```
bsub -n 4 -R "span[ptile=2]" myjob
```

Runs the job on 2 hosts, using 2 processors on each host. Each host may have more than 2 processors available.

```
bsub -n 4 -R "span[ptile=3]" myjob
```

Runs the job on 2 hosts, using 3 processors on the first host and 1 processor on the second host.

```
bsub -n 4 -R "span[ptile=1]" myjob
```

Runs the job on 4 hosts, even though some of the 4 hosts may have more than one processor currently available.

```
bsub -n 4 -R "type==any same[type] span[ptile='!',LINUX:2,SGI:4]" myjob
```

Submits `myjob` to request 4 processors running on 2 hosts of type `LINUX` (2 processors per host), or a single host of type `SGI`, or for other host types, the predefined maximum job slot limit in `lsb.hosts (MXJ)`.

```
bsub -n 16 -R "type==any same[type] span[ptile='!',HP:8,SGI:8,LINUX:2]" myjob
```

Submits `myjob` to request 16 processors on 2 hosts of type `HP` or `SGI` (8 processors per hosts), or on 8 hosts of type `LINUX` (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host types.

```
bsub -n 4 -R "same[model] span[ptile='!',PC1133:4,PC233:2]" myjob
```

Submits `myjob` to request a single host of model `PC1133` (4 processors), or 2 hosts of model `PC233` (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host models.

Specifying parallel job locality at the queue level

The queue may also define the locality for parallel jobs using the `RES_REQ` parameter.

Running Parallel Processes on Homogeneous Hosts

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts some processes from a parallel job may for example, run on Solaris and some on SGI IRIX. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

You can use the `same` section in the resource requirement string to indicate to LSF that processes are to run on one type or model of host. You can also use a custom resource to define the criteria for homogeneous hosts.

Examples

Running all parallel processes on the same host type

```
bsub -n 4 -R"select[type==SGI6 || type==SOL7] same[type]" myjob
```

Allocate 4 processors on the same host type—either SGI IRIX, or Solaris 7, but not both.

Running all parallel processes on the same host type and model

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

Running all parallel processes on hosts in the same high-speed connection group

```
bsub -n 12 -R "select[type==any && (hgconnect==hg1 || hgconnect==hg2 || hgconnect==hg3)] same[hgconnect:type]" myjob
```

For performance reasons, you want to have LSF allocate 12 processors on hosts in high-speed connection group `hg1`, `hg2`, or `hg3`, but not across hosts in `hg1`, `hg2` or `hg3` at the same time. You also want hosts that are chosen to be of the same host type.

Limiting the Number of Processors Allocated

This example reflects a network in which network connections among hosts in the same group are high-speed, and network connections between host groups are low-speed.

In order to specify this, you create a custom resource `hgconnect` in `lsf.shared`.

```
Begin Resource
RESOURCENAME  TYPE      INTERVAL  INCREASING  RELEASE      DESCRIPTION
hgconnect     STRING   ( )        ( )         ( )          (OS release)
...
End Resource
```

In the `lsf.cluster.cluster_name` file, identify groups of hosts that share high-speed connections.

```
Begin ResourceMap
RESOURCENAME  LOCATION
hgconnect     (hg1@[hostA hostB] hg2@[hostD hostE] hg3@[hostF hostG hostX])
End ResourceMap
```

If you want to specify the same resource requirement at the queue-level, define a custom resource in `lsf.shared` as in the previous example, map hosts to high-speed connection groups in `lsf.cluster.cluster_name`, and define the following queue in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = My_test
PRIORITY = 30
NICE = 20
RES_REQ = "select[mem > 1000 && type==any && (hgconnect==hg1 || hgconnect==hg2 || hgconnect=hg3)]same[hgconnect:type]"
DESCRIPTION = either hg1 or hg2 or hg3
End Queue
```

This example allocates processors on hosts that:

- ◆ Have more than 1000 MB in memory
- ◆ Are of the same host type
- ◆ Are in high-speed connection group `hg1` or `hg2` or `hg3`

Limiting the Number of Processors Allocated

Use the `PROCLIMIT` parameter in `lsb.queues` or `lsb.applications` to limit the number of processors that can be allocated to a parallel job.

- ◆ [Syntax](#) on page 478
- ◆ [How PROCLIMIT affects submission of parallel jobs](#) on page 479
- ◆ [Changing PROCLIMIT](#) on page 479
- ◆ [MultiCluster](#) on page 479
- ◆ [Automatic queue selection](#) on page 479
- ◆ [Examples](#) on page 480

Syntax

```
PROCLIMIT = [minimum_limit [default_limit]] maximum_limit
```

All limits must be positive numbers greater than or equal to 1 that satisfy the following relationship:

$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$

You can specify up to three limits in the PROCLIMIT parameter:

If you specify ...	Then ...
One limit	It is the maximum processor limit. The minimum and default limits are set to 1.
Two limits	The first is the minimum processor limit, and the second is the maximum. The default is set equal to the minimum. The minimum must be less than or equal to the maximum.
Three limits	The first is the minimum processor limit, the second is the default processor limit, and the third is the maximum. The minimum must be less than the default and the maximum.

How PROCLIMIT affects submission of parallel jobs

The `-n` option of `bsub` specifies the number of processors to be used by a parallel job, subject to the processor limits of the queue or application profile.

Jobs that specify fewer processors than the minimum PROCLIMIT or more processors than the maximum PROCLIMIT are rejected.

If a default value for PROCLIMIT is specified, jobs submitted without specifying `-n` use the default number of processors. If the queue or application profile has only minimum and maximum values for PROCLIMIT, the number of processors is equal to the minimum value. If only a maximum value for PROCLIMIT is specified, or no PROCLIMIT is specified, the number of processors is equal to 1.

Incorrect processor limits are ignored, and a warning message is displayed when LSF is reconfigured or restarted. A warning message is also logged to the `mbatchd` log file when LSF is started.

Changing PROCLIMIT

If you change the PROCLIMIT parameter, the new processor limit does not affect running jobs. Pending jobs with no processor requirements use the new default PROCLIMIT value. If the pending job does not satisfy the new processor limits, it remains in PENDING state, and the pending reason changes to the following:

```
Job no longer satisfies PROCLIMIT configuration
```

If PROCLIMIT specification is incorrect (for example, too many parameters), a reconfiguration error message is issued. Reconfiguration proceeds and the incorrect PROCLIMIT is ignored.

MultiCluster

Jobs forwarded to a remote cluster are subject to the processor limits of the remote queues. Any processor limits specified on the local cluster are not applied to the remote job.

Automatic queue selection

When you submit a parallel job without specifying a queue name, LSF automatically selects the most suitable queue from the queues listed in the `DEFAULT_QUEUE` parameter in `lsb.params` or the `LSB_DEFAULTQUEUE` environment variable. Automatic queue selection takes into account any maximum and minimum PROCLIMIT values for the queues available for automatic selection.

Limiting the Number of Processors Allocated

If you specify `-n min_proc,max_proc`, but do not specify a queue, the first queue that satisfies the processor requirements of the job is used. If no queue satisfies the processor requirements, the job is rejected.

Example

For example, queues with the following PROCLIMIT values are defined in `lsb.queues`:

- ◆ queueA with PROCLIMIT=1 1 1
- ◆ queueB with PROCLIMIT=2 2 2
- ◆ queueC with PROCLIMIT=4 4 4
- ◆ queueD with PROCLIMIT=8 8 8
- ◆ queueE with PROCLIMIT=16 16 16

In `lsb.params`: `DEFAULT_QUEUE=queueA queueB queueC queueD queueE`

For the following jobs:

```
bsub -n 8 myjob
```

LSF automatically selects queueD to run myjob.

```
bsub -n 5 myjob
```

Job myjob fails because no default queue has the correct number of processors.

Examples

Maximum processor limit

PROCLIMIT is specified in the default queue in `lsb.queues` as:

```
PROCLIMIT = 3
```

The maximum number of processors that can be allocated for this queue is 3.

Example	Description
<code>bsub -n 2 myjob</code>	The job myjob runs on 2 processors.
<code>bsub -n 4 myjob</code>	The job myjob is rejected from the queue because it requires more than the maximum number of processors configured for the queue (3).
<code>bsub -n 2,3 myjob</code>	The job myjob runs on 2 or 3 processors.
<code>bsub -n 2,5 myjob</code>	The job myjob runs on 2 or 3 processors, depending on how many slots are currently available on the host.
<code>bsub myjob</code>	No default or minimum is configured, so the job myjob runs on 1 processor.

Minimum and maximum processor limits

PROCLIMIT is specified in `lsb.queues` as:

```
PROCLIMIT = 3 8
```

The minimum number of processors that can be allocated for this queue is 3 and the maximum number of processors that can be allocated for this queue is 8.

Example	Description
<code>bsub -n 5 myjob</code>	The job <code>myjob</code> runs on 5 processors.
<code>bsub -n 2 myjob</code>	The job <code>myjob</code> is rejected from the queue because the number of processors requested is less than the minimum number of processors configured for the queue (3).
<code>bsub -n 4,5 myjob</code>	The job <code>myjob</code> runs on 4 or 5 processors.
<code>bsub -n 2,6 myjob</code>	The job <code>myjob</code> runs on 3 to 6 processors.
<code>bsub -n 4,9 myjob</code>	The job <code>myjob</code> runs on 4 to 8 processors.
<code>bsub myjob</code>	The default number of processors is equal to the minimum number (3). The job <code>myjob</code> runs on 3 processors.

Minimum, default, and maximum processor limits

PROCLIMIT is specified in `lsb.queues` as:

```
PROCLIMIT = 4 6 9
```

- ◆ Minimum number of processors that can be allocated for this queue is 4
- ◆ Default number of processors for the queue is 6
- ◆ Maximum number of processors that can be allocated for this queue is 9

Example	Description
<code>bsub myjob</code>	Because a default number of processors is configured, the job <code>myjob</code> runs on 6 processors.

Reserving Processors

About processor reservation

When parallel jobs have to compete with sequential jobs for job slots, the slots that become available are likely to be taken immediately by a sequential job. Parallel jobs need multiple job slots to be available before they can be dispatched. If the cluster is always busy, a large parallel job could be pending indefinitely. The more processors a parallel job requires, the worse the problem is.

Processor reservation solves this problem by reserving job slots as they become available, until there are enough reserved job slots to run the parallel job.

You might want to configure processor reservation if your cluster has a lot of sequential jobs that compete for job slots with parallel jobs.

How processor reservation works

Processor reservation is disabled by default.

If processor reservation is enabled, and a parallel job cannot be dispatched because there are not enough job slots to satisfy its minimum processor requirements, the job slots that are currently available will be reserved and accumulated.

A reserved job slot is unavailable to any other job. To avoid deadlock situations in which the system reserves job slots for multiple parallel jobs and none of them can acquire sufficient resources to start, a parallel job will give up all its reserved job slots if it has not accumulated enough to start within a specified time. The

Reserving Memory for Pending Parallel Jobs

reservation time starts from the time the first slot is reserved. When the reservation time expires, the job cannot reserve any slots for one scheduling cycle, but then the reservation process can begin again.

If you specify first execution host candidates at the job or queue level, LSF will try to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it will not reserve slots on any other hosts.

Configure processor reservation

- 1 To enable processor reservation, set `SLOT_RESERVE` in `lsb.queues` and specify the reservation time (a job cannot hold any reserved slots after its reservation time expires).

Syntax

```
SLOT_RESERVE=MAX_RESERVE_TIME[n].
```

where *n* is an integer by which to multiply `MBD_SLEEP_TIME`.

`MBD_SLEEP_TIME` is defined in `lsb.params`; the default value is 60 seconds.

Example

```
Begin Queue
.
PJOB_LIMIT=1
SLOT_RESERVE = MAX_RESERVE_TIME[5]
.
End Queue
```

In this example, if `MBD_SLEEP_TIME` is 60 seconds, a job can reserve job slots for 5 minutes. If `MBD_SLEEP_TIME` is 30 seconds, a job can reserve job slots for 5 * 30 = 150 seconds, or 2.5 minutes.

Viewing information about reserved job slots

Reserved slots can be displayed with the `bjobs` command. The number of reserved slots can be displayed with the `bqueues`, `bhosts`, `bhpart`, and `busers` commands. Look in the `RSV` column.

Reserving Memory for Pending Parallel Jobs

By default, the `usage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available, until the total required memory specified on the `usage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

Unlike slot reservation, which only applies to parallel jobs, memory reservation applies to both sequential and parallel jobs.

Configuring memory reservation for pending parallel jobs

Use the `RESOURCE_RESERVE` parameter in `lsb.queues` to reserve host memory for pending jobs, as described in [Memory Reservation for Pending Jobs](#) on page 390.

lsb.queues

- 1 Set the `RESOURCE_RESERVE` parameter in a queue defined in `lsb.queues`.

The `RESOURCE_RESERVE` parameter overrides the `SLOT_RESERVE` parameter. If both `RESOURCE_RESERVE` and `SLOT_RESERVE` are defined in the same queue, job slot reservation and memory reservation are enabled and an error is displayed when the cluster is reconfigured. `SLOT_RESERVE` is ignored. Backfill on memory may still take place.

The following queue enables both memory reservation and backfill in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Enable per-slot memory reservation

By default, memory is reserved for parallel jobs on a per-host basis. For example, by default, the command:

```
bsub -n 4 -R "rusage[mem=500]" -q reservation myjob
```

requires the job to reserve 500 MB on each host where the job runs.

- 1 To enable per-slot memory reservation, define `RESOURCE_RESERVE_PER_SLOT=y` in `lsb.params`. In this example, if per-slot reservation is enabled, the job must reserve 500 MB of memory for each job slot ($4 * 500 = 2$ GB) on the host in order to run.

Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots

By default, a reserved job slot cannot be used by another job. To make better use of resources and improve performance of LSF, you can configure backfill scheduling.

About backfill scheduling

Backfill scheduling allows other jobs to use the reserved job slots, as long as the other jobs will not delay the start of another job. Backfilling, together with processor reservation, allows large parallel jobs to run while not underutilizing resources.

In a busy cluster, processor reservation helps to schedule large parallel jobs sooner. However, by default, reserved processors remain idle until the large job starts. This degrades the performance of LSF because the reserved resources are idle while jobs are waiting in the queue.

Backfill scheduling allows the reserved job slots to be used by small jobs that can run and finish before the large job starts. This improves the performance of LSF because it increases the utilization of resources.

How backfilling works

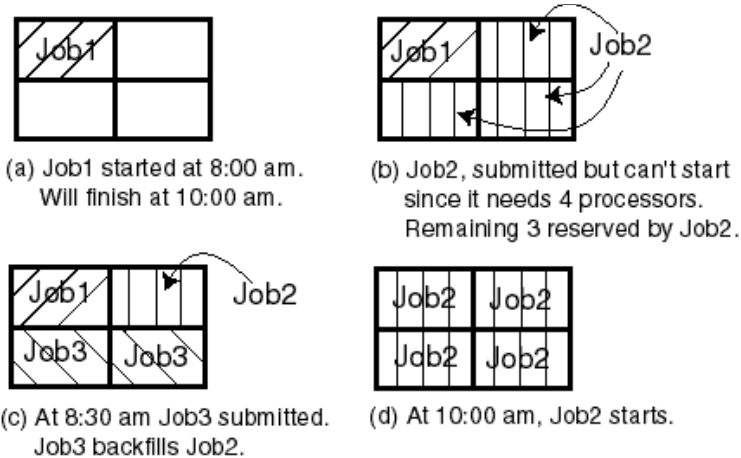
For backfill scheduling, LSF assumes that a job will run until its run limit expires. Backfill scheduling works most efficiently when all the jobs in the cluster have a run limit.

Since jobs with a shorter run limit have more chance of being scheduled as backfill jobs, users who specify appropriate run limits in a backfill queue will be rewarded by improved turnaround time.

Once the big parallel job has reserved sufficient job slots, LSF calculates the start time of the big job, based on the run limits of the jobs currently running in the reserved slots. LSF cannot backfill if the big job is waiting for a job that has no run limit defined.

If LSF can backfill the idle job slots, only jobs with run limits that expire before the start time of the big job will be allowed to use the reserved job slots. LSF cannot backfill with a job that has no run limit.

Example



In this scenario, assume the cluster consists of a 4-CPU multiprocessor host.

- 1 A sequential job (*job1*) with a run limit of 2 hours is submitted and gets started at 8:00 am (figure a).
- 2 Shortly afterwards, a parallel job (*job2*) requiring all 4 CPUs is submitted. It cannot start right away because *job1* is using one CPU, so it reserves the remaining 3 processors (figure b).
- 3 At 8:30 am, another parallel job (*job3*) is submitted requiring only two processors and with a run limit of 1 hour. Since *job2* cannot start until 10:00am (when *job1* finishes), its reserved processors can be backfilled by *job3* (figure c). Therefore *job3* can complete before *job2*'s start time, making use of the idle processors.
- 4 *Job3* will finish at 9:30am and *job1* at 10:00am, allowing *job2* to start shortly after 10:00am. In this example, if *job3*'s run limit was 2 hours, it would not be able to backfill *job2*'s reserved slots, and would have to run after *job2* finishes.

Limitations

- ◆ A job will not have an estimated start time immediately after `mbatchd` is reconfigured.

Backfilling and job slot limits

A backfill job borrows a job slot that is already taken by another job. The backfill job will not run at the same time as the job that reserved the job slot first. Backfilling can take place even if the job slot limits for a host or processor have been reached. Backfilling cannot take place if the job slot limits for users or queues have been reached.

Configuring backfill scheduling

Backfill scheduling is enabled at the queue level. Only jobs in a backfill queue can backfill reserved job slots. If the backfill queue also allows processor reservation, then backfilling can occur among jobs within the same queue.

Configure a backfill queue

- 1 To configure a backfill queue, define `BACKFILL` in `lsb.queues`.
- 2 Specify `Y` to enable backfilling. To disable backfilling, specify `N` or blank space.

Example

```
BACKFILL=Y
```

Enforcing run limits

Backfill scheduling requires all jobs to specify a duration. If you specify a run time limit using the command line `bsub -W` option or by defining the `RUNLIMIT` parameter in `lsb.queues` or `lsb.applications`, LSF uses that value as a hard limit and terminates jobs that exceed the specified duration. Alternatively, you can specify an estimated duration by defining the `RUNTIME` parameter in `lsb.applications`. LSF uses the `RUNTIME` estimate for scheduling purposes only, and does not terminate jobs that exceed the `RUNTIME` duration.

Backfill scheduling works most efficiently when all the jobs in a cluster have a run limit specified at the job level (`bsub -W`). You can use the external submission executable, `esub`, to make sure that all users specify a job-level run limit.

Otherwise, you can specify ceiling and default run limits at the queue level (`RUNLIMIT` in `lsb.queues`) or application level (`RUNLIMIT` in `lsb.applications`).

View information about job start time

1 Use `bjobs -l` to view the estimated start time of a job.

Using backfill on memory

If `BACKFILL` is configured in a queue, and a run limit is specified with `-W` on `bsub` or with `RUNLIMIT` in the queue, backfill jobs can use the accumulated memory reserved by the other jobs, as long as the backfill job can finish before the predicted start time of the jobs with the reservation.

Unlike slot reservation, which only applies to parallel jobs, backfill on memory applies to sequential and parallel jobs.

The following queue enables both memory reservation and backfill on memory in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Examples of memory reservation and backfill on memory

lsb.queues

The following queues are defined in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue

Begin Queue
QUEUE_NAME = backfill
DESCRIPTION = For backfill scheduling
PRIORITY = 30
BACKFILL = y
End Queue
```

lsb.params

Per-slot memory reservation is enabled by `RESOURCE_RESERVE_PER_SLOT=y` in `lsb.params`.

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory. The first three jobs will run for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job will start running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements will get the same result.

Job 3:

Submitting a third job with same requirements will reserve one job slot, and reserve all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myjob4
```

The job will keep pending, since memory is reserved by job 3 and it will run longer than job 1 and job 2.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myjob5
```

The job will start running. It uses one free slot and memory reserved by job 3. If the job does not finish in 100 minutes, it will be killed by LSF automatically.

Job 6:

```
bsub -W 100 -q backfill -R "rusage[mem=300]" myjob6
```

The job will keep pending with no resource reservation because it cannot get enough memory from the memory reserved by job 3.

Job 7:

```
bsub -W 100 -q backfill myjob7
```

The job will start running. LSF assumes it does not require any memory and enough job slots are free.

Parallel jobs

Each process of a parallel job requires 100 MB memory, and each parallel job needs 4 cpus. The first two of the following parallel jobs will run for 300 minutes.

Job 1:

```
bsub -W 300 -n 4 -R "rusage[mem=100]" -q reservation myJob1
```

The job will start running and use 4 slots and get 400MB memory.

Job 2:

Submitting a second job with same requirements will get the same result.

Job 3:

Submitting a third job with same requirements will reserve 2 slots, and reserve all 200 MB of available memory, assuming no other applications are running outside of LSF.

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myJob4
```

The job will keep pending since all available memory is already reserved by job 3. It will run longer than job 1 and job 2, so no backfill happens.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myJob5
```

This job will start running. It can backfill the slot and memory reserved by job 3. If the job does not finish in 100 minutes, it will be killed by LSF automatically.

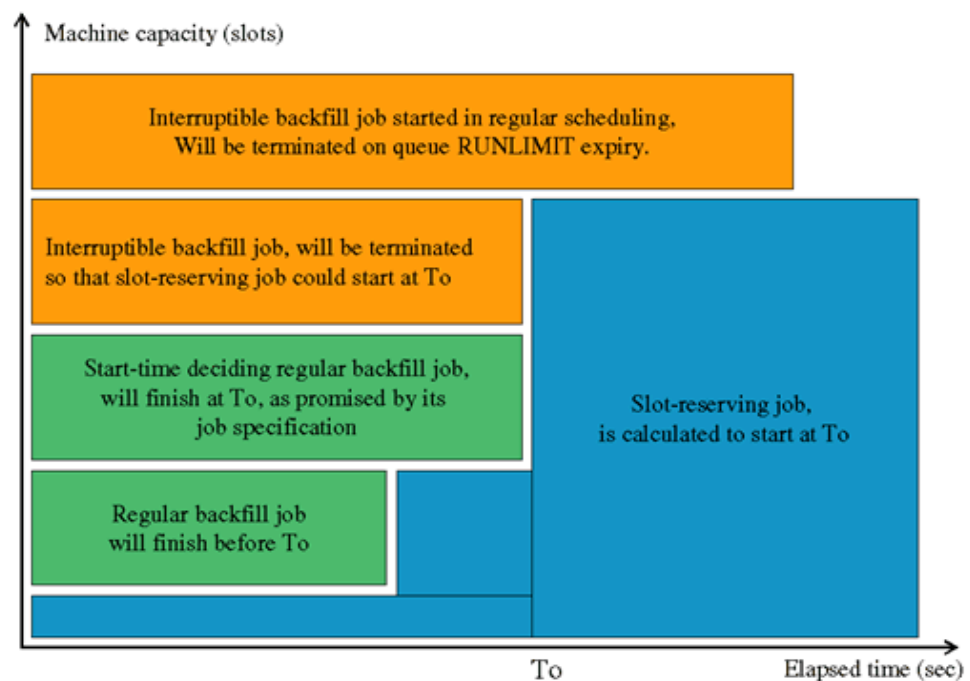
Using interruptible backfill

Interruptible backfill scheduling can improve cluster utilization by allowing reserved job slots to be used by low priority small jobs that will be terminated when the higher priority large jobs are about to start.

An interruptible backfill job:

- ◆ Starts as a regular job and is killed when it exceeds the queue runtime limit, or
- ◆ Is started for backfill whenever there is a backfill time slice longer than the specified minimal time, and killed before the slot-reservation job is about to start. This applies to compute-intensive serial or single-node parallel jobs that can run a long time, yet be able to checkpoint or resume from an arbitrary computation point.

Resource allocation diagram



Job life cycle

- 1 Jobs are submitted to a queue configured for interruptible backfill. The job runtime requirement is ignored.
- 2 Job is scheduled as either regular job or backfill job.
- 3 The queue runtime limit is applied to the regularly scheduled job.
- 4 In backfill phase, the job is considered for run on any reserved resource, which duration is longer than the minimal time slice configured for the queue. The job runtime limit is set in such way, that the job will release the resource before it is needed by the slot reserving job.
- 5 The job runs in a regular manner. It is killed upon reaching its runtime limit, and requeued for the next run. Requeueing must be explicitly configured in the queue.

Assumptions and limitations

- ◆ The interruptible backfill job will hold the slot-reserving job start until its calculated start time, in the same way as a regular backfill job. The interruptible backfill job is killed when its run limit expires.
- ◆ Killing other running jobs prematurely does not affect the calculated run limit of an interruptible backfill job. Slot-reserving jobs will not start sooner.
- ◆ While the queue is checked for the consistency of interruptible backfill, backfill and runtime specifications, the requeue exit value clause is not verified, nor executed automatically. Configure requeue exit values according to your site policies.
- ◆ In LSF MultiCluster, `bhist` does not display interruptible backfill information for remote clusters.
- ◆ A migrated job belonging to an interruptible backfill queue is migrated as if `LSB_MIG2PEND` is set.

Configure an interruptible backfill queue

- 1 Configure `INTERRUPTIBLE_BACKFILL=seconds` in the lowest priority queue in the cluster. There can only be one interruptible backfill queue in the cluster.

Specify the minimum number of seconds for the job to be considered for backfilling. This minimal time slice depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created if a site has jobs of distinctively different classes.

For example:

```
Begin Queue
QUEUE_NAME = background
# REQUEUE_EXIT_VALUES (set to whatever needed)
DESCRIPTION = Interruptible Backfill queue
BACKFILL = Y
INTERRUPTIBLE_BACKFILL = 1
RUNLIMIT = 10
PRIORITY = 1
End Queue
```

Interruptible backfill is disabled if `BACKFILL` and `RUNLIMIT` are not configured in the queue.

The value of `INTERRUPTIBLE_BACKFILL` is the minimal time slice in seconds for a job to be considered for backfill. The value depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created for different classes of jobs.

`BACKFILL` and `RUNLIMIT` must be configured in the queue.

`RUNLIMIT` corresponds to a maximum time slice for backfill, and should be configured so that the wait period for the new jobs submitted to the queue is acceptable to users. 10 minutes of runtime is a common value.

Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots

You should configure `REQUEUE_EXIT_VALUES` for the queue so that resubmission is automatic. In order to terminate completely, jobs must have specific exit values:

- ◆ If jobs are checkpointable, use their checkpoint exit value.
- ◆ If jobs periodically save data on their own, use the `SIGTERM` exit value.

View the run limits for interruptible backfill jobs (bjobs and bhist)

- 1 Use `bjobs` to display the run limit calculated based on the configured queue-level run limit.

For example, the interruptible backfill queue `lazy` configures `RUNLIMIT=60`:

```
bjobs -l 135
Job <135>, User <user1>, Project <default>, Status <RUN>, Queue <lazy>, Command
    <myjob>
Mon Nov 21 11:49:22: Submitted from host <hostA>, CWD <${HOME}/H
    PC/jobs>;

RUNLIMIT
59.5 min of hostA
Mon Nov 21 11:49:26: Started on <hostA>, Execution Home </home
    /user1>, Execution CWD </home/user1/HPC/jobs>;
```

- 2 Use `bhist` to display job-level run limit if specified.

For example, job 135 was submitted with a run limit of 3 hours:

```
bsub -n 1 -q lazy -W 3:0 myjob
Job <135> is submitted to queue <lazy>.
bhist displays the job-level run limit:
```

```
bhist -l 135
Job <135>, User <user1>, Project <default>, Command <myjob>
Mon Nov 21 11:49:22: Submitted from host <hostA>, to Queue <la
    zy>, CWD <${HOME}/HPC/jobs>;

RUNLIMIT
180.0 min of hostA
Mon Nov 21 11:49:26: Dispatched to <hostA>;
Mon Nov 21 11:49:26: Starting (Pid 2746);
Mon Nov 21 11:49:27: Interruptible backfill runtime limit is 59.5 minutes;
Mon Nov 21 11:49:27: Running with execution home </home/user1>, Execution CWD
...

```

Displaying available slots for backfill jobs

The `bslots` command displays slots reserved for parallel jobs and advance reservations. The available slots are not currently used for running jobs, and can be used for backfill jobs. The available slots displayed by `bslots` are only a snapshot of the slots currently not in use by parallel jobs or advance reservations. They are not guaranteed to be available at job submission.

By default, `bslots` displays all available slots, and the available run time for those slots. When no reserved slots are available for backfill, `bslots` displays "No reserved slots available."

The backfill window calculation is based on the snapshot information (current running jobs, slot reservations, advance reservations) obtained from `mbatchd`.

The backfill window displayed can serve as reference for submitting backfillable jobs. However, if you have specified extra resource requirements or special submission options, it does not insure that submitted jobs will be scheduled/dispatched successfully.

`bslots -R` only supports the `select` resource requirement string. Other resource requirement selections are not supported.

If the available backfill window has no run time limit, its length is displayed as UNLIMITED.

Examples

Display all available slots for backfill jobs:

```
bslots
SLOTS RUNTIME
1  UNLIMITED
3  1 hour 30 minutes
5  1 hour 0 minutes
7  45 minutes
15 40 minutes
18 30 minutes
20 20 minutes
```

Display available slots for backfill jobs requiring 15 slots or more:

```
bslots -n 15
SLOTS RUNTIME
15 40 minutes
18 30 minutes
20 20 minutes
```

Display available slots for backfill jobs requiring a run time of 30 minutes or more:

```
bslots -W 30
SLOTS RUNTIME
3  1 hour 30 minutes
5  1 hour 0 minutes
7  45 minutes
15 40 minutes
18 30 minutes
bslots -W 2:45
```

No reserved slots available.

```
bslots -n 15 -W 30
```

SLOTS RUNTIME

15 40 minutes

18 30 minutes

Display available slots for backfill jobs requiring a host with more than 500 MB of memory:

```
bslots -R "mem>500"
```

SLOTS RUNTIME

7 45 minutes

15 40 minutes

Display the host names with available slots for backfill jobs:

```
bslots -l
```

SLOTS: 15

RUNTIME: 40 minutes

HOSTS: 1*hostB 1*hostE 3*hostC ...
3*hostZ

SLOTS: 15

RUNTIME: 30 minutes

HOSTS: 2*hostA 1*hostB 3*hostC ...
1*hostX

Submitting backfill jobs according to available slots

- 1 Use `bslots` to display job slots available for backfill jobs.
- 2 Submit a job to a backfill queue. Specify a runtime limit and the number of processors required that are within the availability shown by `bslots`.

Submitting a job according to the backfill slot availability shown by `bslots` does not guarantee that the job will be backfilled successfully. The slots may not be available by the time job is actually scheduled, or the job cannot be dispatched because other resource requirements are not satisfied.

Parallel Fairshare

LSF can consider the number of CPUs when using fairshare scheduling with parallel jobs.

If the job is submitted with `bsub -n`, the following formula is used to calculate dynamic priority:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{number_CPUs} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR})$$

where *number_CPUs* is the number of CPUs used by the job.

Configure parallel fairshare

To configure parallel fairshare so that the number of CPUs is considered when calculating dynamic priority for queue-level user-based fairshare:

NOTE: `LSB_NCPU_ENFORCE` does not apply to host-partition user-based fairshare. For host-partition user-based fairshare, the number of CPUs is automatically considered.

- 1 Configure fairshare at the queue level as indicated in [Fairshare Scheduling](#) on page 295.
- 2 To enable parallel fairshare, set the parameter `LSB_NCPU_ENFORCE=1` in `lsf.conf`.
- 3 To make your changes take effect, use the following commands to restart all LSF daemons:


```
# lsadmin reconfig
# lsadmin resrestart all
# badmin hrestart all
# badmin mbdrestart
```

How Deadline Constraint Scheduling Works For Parallel Jobs

For information about deadline constraint scheduling, see [Using Deadline Constraint Scheduling](#) on page 275. Deadline constraint scheduling is enabled by default.

If deadline constraint scheduling is enabled and a parallel job has a CPU limit but no run limit, LSF considers the number of processors when calculating how long the job will take.

LSF assumes that the minimum number of processors will be used, and that they will all be the same speed as the candidate host. If the job cannot finish under these conditions, LSF will not place the job.

The formula is:

$$(\text{deadline time} - \text{current time}) > (\text{CPU limit on candidate host} / \text{minimum number of processors})$$

Optimized Preemption of Parallel Jobs

You can configure preemption for parallel jobs to reduce the number of jobs suspended in order to run a large parallel job.

When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job.

The `PREEMPT_FOR` parameter in `lsb.params` with the `MINI_JOB` keyword enables the optimized preemption of parallel jobs, so LSF preempts fewer of the low-priority parallel jobs.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs.

How optimized preemption works

When you run many parallel jobs in your cluster, and parallel jobs preempt other parallel jobs, you can enable a feature to optimize the preemption mechanism among parallel jobs.

By default, LSF can over-preempt parallel jobs. When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job. The optimized preemption mechanism reduces the number of jobs that are preempted.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs. Enabling or disabling this feature has no effect on the scheduling of jobs that require only a single processor.

Configure optimized preemption

-
- 1 Use the `PREEMPT_FOR` parameter in `lsb.params` and specify the keyword `MINI_JOB` to configure optimized preemption at the cluster level.

If the parameter is already set, the `MINI_JOB` keyword can be used along with other keywords; the other keywords do not enable or disable the optimized preemption mechanism.

Submitting Jobs Using JSDL

Contents

- ◆ [Why Use JSDL?](#) on page 495
- ◆ [Using JSDL Files with LSF](#) on page 495
- ◆ [Collecting resource values using elim.jsdl](#) on page 504

Why Use JSDL?

The Job Submission Description Language (JSDL) provides a convenient format for describing job requirements. You can save a set of job requirements in a JSDL XML file, and then reuse that file as needed to submit jobs to LSF.

For detailed information about JSDL, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Using JSDL Files with LSF

LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.

The following sections describe how LSF supports the use of JSDL files for job submission.

Where to find the JSDL schema files

The JSDL schema (`jsdl.xsd`), the POSIX extension (`jsdl-posix.xsd`), and the LSF extension (`jsdl-lsf.xsd`) are located in the `LSF_LIBDIR` directory.

Supported JSDL and POSIX extension elements

The following table maps the supported JSDL standard and POSIX extension elements to LSF submission options.

NOTE: For information about how to specify JSDL element types such as range values, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Table 1: Supported JSDL and POSIX extension elements

Element	bsub Option	Description	Example
Job Structure Elements			
JobDefinition	N/A	Root element of the JSDL document. Contains the mandatory child element JobDescription.	<code><JobDefinition> <JobDescription> ... </JobDescription> </JobDefinition></code>
JobDescription	-P	High-level container element that holds more specific job description elements.	
Job Identity Elements			
JobName	-J	String used to name the job.	<code><jsd1:JobName>myjob</jsdl:JobName></code>
JobProject	-P	String that specifies the project to which the job belongs.	<code><jsd1:JobProject>myproject </jsdl:JobProject></code>
Application Elements			
Application	N/A	High-level container element that holds more specific application definition elements.	
ApplicationName	-app	String that defines the name of an application profile defined in <code>lsb.applications</code> .	<code><jsd1:Application> <jsd1:ApplicationName>ApplicationX </jsdl:ApplicationName> </jsdl:Application></code>
ApplicationVersion	-app	String that defines the version of the application defined in <code>lsb.applications</code> .	<code><jsd1:Application> <jsd1:ApplicationName> ApplicationX</jsdl:ApplicationName> <jsd1:ApplicationVersion>5.5 </jsdl:ApplicationVersion> ... </jsdl:Application></code>
Resource Elements			
CandidateHosts	-m	Complex type element that specifies the set of named hosts that can be selected to run the job.	<code><jsd1:CandidateHosts> <jsd1:HostName>host1</jsdl:HostName> <jsd1:HostName>host2</jsdl:HostName> </jsdl:CandidateHosts></code>
HostName	-m	Contains a single name of a host or host group. See the previous example (CandidateHosts).	

Element	bsub Option	Description	Example
ExclusiveExecution	-x	Boolean that designates whether the job must have exclusive access to the resources it uses.	<code><jsd1:ExclusiveExecution>>true </jsd1:ExclusiveExecution></code>
OperatingSystemName	-R	A token type that contains the operating system name. LSF will use the external resource "osname."	<code><jsd1:OperatingSystemName>LINUX </jsd1:OperatingSystemName></code>
OperatingSystemVersion	-R	A token type that contains the operating system version. LSF will use the external resource "osver."	<code><jsd1:OperatingSystemVersion>5.7 </jsd1:OperatingSystemVersion></code>
CPUArchitectureName	-R	Token that specifies the CPU architecture required by the job in the execution environment. LSF will use the external resource "cpuarch."	<code><jsd1:CPUArchitectureName>sparc </jsd1:CPUArchitectureName></code>
IndividualCPUSpeed	-R	Range value that specifies the speed of each CPU required by the job in the execution environment, in Hertz (Hz). LSF will use the external resource "cpuspeed."	<code><jsd1:IndividualCPUSpeed> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualCPUSpeed></code>
IndividualCPUCount	-n	Range value that specifies the number of CPUs for each resource.	<code><jsd1:IndividualCPUCount> <jsd1:exact>2.0</jsd1:exact> </jsd1:IndividualCPUCount></code>
IndividualPhysicalMemory	-R	Range value that specifies the amount of physical memory required on each resource, in bytes.	<code><jsd1:IndividualPhysicalMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualPhysicalMemory></code>
IndividualVirtualMemory	-R	Range value that specifies the amount of virtual memory required for each resource, in bytes.	<code><jsd1:IndividualVirtualMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualVirtualMemory></code>

Using JSDL Files with LSF

Element	bsub Option	Description	Example
IndividualNetworkBandwidth	-R	Range value that specifies the bandwidth requirements of each resource, in bits per second (bps). LSF will use the external resource "bandwidth."	<code><jsd1:IndividualNetworkBandwidth> <jsd1:LowerBoundedRange>104857600.0 </jsdl:LowerBoundedRange> </jsdl:IndividualNetworkBandwidth></code>
TotalCPUCount	-n	Range value that specifies the total number of CPUs required for the job.	<code><jsd1:TotalCPUCount><jsd1:exact>2.0 </jsdl:exact></jsdl:TotalCPUCount></code>
TotalPhysicalMemory	-R	Range value that specifies the required amount of physical memory for all resources for the job, in bytes.	<code><jsd1:TotalPhysicalMemory> <jsd1:LowerBoundedRange> 10737418240.0 </jsdl:LowerBoundedRange> </jsdl:TotalPhysicalMemory></code>
TotalVirtualMemory	-R	Range value that specifies the required amount of virtual memory for the job, in bytes.	<code><jsd1:TotalVirtualMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsdl:LowerBoundedRange> </jsdl:TotalVirtualMemory></code>
TotalResourceCount	-n	Range value that specifies the total number of resources required by the job.	<code><jsd1:Resources>... <jsd1:TotalResourceCount> <jsd1:exact>5.0</jsdl:exact> </jsdl:TotalResourceCount></code>
Data Staging Elements			
FileName	-f	String that specifies the local name of the file or directory on the execution host. For a directory, you must specify the relative path.	<code><jsd1:DataStaging><jsd1:FileName> job1/input/control.txt </jsdl:FileName> ...</jsdl:DataStaging></code>
CreationFlag	-f	Specifies whether the file created on the local execution system will overwrite or append to an existing file.	<code><jsd1:DataStaging> <jsd1:CreationFlag>overwrite </jsdl:CreationFlag> ...</jsdl:DataStaging></code>
Source	N/A	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file will be staged in before the job is executed. See the example for the Target element.	

Element	bsub Option	Description	Example
URI	-f	Specifies the location used to stage in (Source) or stage out (Target) a file. For use with LSF, the URI must be a file path only, without a protocol.	
Target	N/A	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file will be staged out after the job is executed.	<pre><jsd1:DataStaging><jsd1:Source> <jsd1:URI>//input/myjobs/control.txt </jsdl:URI></jsdl:Source> <jsd1:Target><jsd1:URI> //output/myjobs/control.txt </jsdl:URI></jsdl:Target> ...</jsdl:DataStaging></pre>
POSIX Extension Elements			
Executable	N/A	String that specifies the command to execute.	<pre><jsd1-posix:Executable>myscript </jsdl-posix:Executable></pre>
Argument	N/A	Constrained normalized string that specifies an argument for the application or command.	<pre><jsd1-posix:Argument>10 </jsdl-posix:Argument></pre>
Input	-l	String that specifies the Standard Input for the command.	<pre>...<jsd1-posix:Input>input.txt </jsdl-posix:Input>...</pre>
Output	-o	String that specifies the Standard Output for the command.	<pre>...<jsd1-posix:Output>output.txt </jsdl-posix:Output>...</pre>
Error	-e	String that specifies the Standard Error for the command.	<pre>...<jsd1-posix:Error>error.txt </jsdl-posix:Error>...</pre>
WorkingDirectory	N/A	String that specifies the starting directory required for job execution. If no directory is specified, LSF sets the starting directory on the execution host to the current working directory on the submission host. If the current working directory is not accessible on the execution host, LSF runs the job in the /tmp directory on the execution host.	<pre>...<jsd1-posix:WorkingDirectory> /home</jsdl-posix:WorkingDirectory> ...</pre>

Using JSDL Files with LSF

Element	bsub Option	Description	Example
Environment	N/A	Specifies the name and value of an environment variable defined for the job in the execution environment. LSF maps the JSDL element definitions to the matching LSF environment variables.	<code><jsd1-posix:Environment name="SHELL"> /bin/bash</jssl-posix:Environment></code>
WallTimeLimit	-W	Positive integer that specifies the soft limit on the duration of the application's execution, in seconds.	<code><jsd1-posix:WallTimeLimit>60 </jssl-posix:WallTimeLimit></code>
FileSizeLimit	-F	Positive integer that specifies the maximum size of any file associated with the job, in bytes.	<code><jsd1-posix:FileSizeLimit>1073741824 </jssl-posix:FileSizeLimit></code>
CoreDumpLimit	-C	Positive integer that specifies the maximum size of core dumps a job may create, in bytes.	<code><jsd1-posix:CoreDumpLimit>0 </jssl-posix:CoreDumpLimit></code>
DataSegmentLimit	-D	Positive integer that specifies the maximum data segment size, in bytes.	<code><jsd1-posix:DataSegmentLimit>32768 </jssl-posix:DataSegmentLimit></code>
MemoryLimit	-M	Positive integer that specifies the maximum amount of physical memory that the job can use during execution, in bytes.	<code><jsd1-posix:MemoryLimit>67108864 </jssl-posix:MemoryLimit></code>
StackSizeLimit	-S	Positive integer that specifies the maximum size of the execution stack for the job, in bytes.	<code><jsd1-posix:StackSizeLimit>1048576 </jssl-posix:StackSizeLimit></code>
CPUTimeLimit	-c	Positive integer that specifies the number of CPU time seconds a job can consume before a SIGXCPU signal is sent to the job.	<code><jsd1-posix:CPUTimeLimit>30 </jssl-posix:CPUTimeLimit></code>

Element	bsub Option	Description	Example
ProcessCountLimit	-p	Positive integer that specifies the maximum number of processes the job can spawn.	<code><jsd1-posix:ProcessCountLimit>8</jsdl-posix:ProcessCountLimit></code>
VirtualMemoryLimit	-v	Positive integer that specifies the maximum amount of virtual memory the job can allocate, in bytes.	<code><jsd1-posix:VirtualMemoryLimit>134217728</jsdl-posix:VirtualMemoryLimit></code>
ThreadCountLimit	-T	Positive integer that specifies the number of threads the job can create.	<code><jsd1-posix:ThreadCountLimit>8</jsdl-posix:VirtualMemoryLimit></code>

LSF extension elements

To use all available LSF features, add the elements described in the following table to your JSDL file.

Table 2: LSF extension elements

Element	bsub Option	Description
SchedulerParams	N/A	Complex type element that specifies various scheduling parameters (starting with Queue and ending with JobGroup).
<pre> <jsd1-lsf:SchedulerParams> <jsd1-lsf:Queue>normal</jsdl-lsf:Queue> <jsd1-lsf:ResourceRequirements>"select[swp>15 && hpux] order[ut]" </jsdl-lsf:ResourceRequirements> <jsd1-lsf:Start>12:06:09:55</jsdl-lsf:Start> <jsd1-lsf:Deadline>8:22:15:50</jsdl-lsf:Deadline> <jsd1-lsf:ReservationID>"user1#0"</jsdl-lsf:ReservationID> <jsd1-lsf:Dependencies>'done myjob1'</jsdl-lsf:Dependencies> <jsd1-lsf:Rerunnable>true</jsdl-lsf:Rerunnable> <jsd1-lsf:UserPriority>3</jsdl-lsf:UserPriority> <jsd1-lsf:ServiceClass>platinum</jsdl-lsf:ServiceClass> <jsd1-lsf:Group>sysadmin</jsdl-lsf:Group> <jsd1-lsf:ExternalScheduler></jsdl-lsf:ExternalScheduler><jsd1-lsf:Hold>true </jsdl-lsf:Hold> <jsd1-lsf:JobGroup>/risk_group/portfolio1/current</jsdl-lsf:JobGroup> </jsdl-lsf:SchedulerParams> </pre>		
Queue	-q	String that specifies the queue in which the job will run.
ResourceRequirements	-R	String that specifies one or more resource requirements of the job. Multiple rusage sections are not supported.
Start	-b	String that specifies the earliest time that the job can start.
Deadline	-t	String that specifies the job termination deadline.
ReservationID	-U	String that specifies the reservation ID returned when you use <code>brsvadd</code> to add a reservation.
Dependencies	-w	String that specifies a dependency expression. LSF will not run your job unless the dependency expression evaluates to TRUE.
Rerunnable	-r	Boolean value that specifies whether to reschedule a job on another host if the execution host becomes unavailable while the job is running.

Using JSDL Files with LSF

Element	bsub Option	Description
UserPriority	-sp	Positive integer that specifies the user-assigned job priority. This allows users to order their own jobs within a queue.
ServiceClass	-sla	String that specifies the service class where the job is to run.
Group	-G	String that associates the job with the specified group for fairshare scheduling.
ExternalScheduler	-ext [sched]	String used to set application-specific external scheduling options for the job.
Hold	-H	Boolean value that tells LSF to hold the job in the PSUSP state when the job is submitted. The job will not be scheduled until you tell the system to resume the job.
JobGroup	-g	String that specifies the job group to which the job is submitted.
NotificationParams	N/A	Complex type element that tells LSF when and where to send notification emails for a job. See the following example: <pre><jSDL-lsf:NotificationParams> <jSDL-lsf:NotifyAtStart>true</jSDL-lsf:NotifyAtStart> <jSDL-lsf:NotifyAtFinish>true</jSDL-lsf:NotifyAtFinish> <jSDL-lsf:NotificationEmail>-u user10</jSDL-lsf:NotificationEmail> </jSDL-lsf:NotificationParams></pre>
NotifyAtStart	-B	Boolean value that tells LSF to notify the user who submitted the job that the job has started.
NotifyAtFinish	-N	Boolean value that tells LSF to to notify the user who submitted the job that the job has finished.
NotificationEmail	-u	String that specifies the user who will receive notification emails.
RuntimeParams	N/A	Complex type element that contains values for LSF runtime parameters. <pre><jSDL-lsf:RuntimeParams> <jSDL-lsf:Interactive>-ls</jSDL-lsf:Interactive> <jSDL-lsf:Block>true</jSDL-lsf:Block> <jSDL-lsf:PreExec>myscript</jSDL-lsf:PreExec> <jSDL-lsf:Checkpoint>myjobs/checkpointdir</jSDL-lsf:Checkpoint> <jSDL-lsf:LoginShell>/csh</jSDL-lsf:LoginShell> <jSDL-lsf:SignalJob>18</jSDL-lsf:SignalJob> <jSDL-lsf:WarningAction>'URG'</jSDL-lsf:WarningAction> <jSDL-lsf:WarningTime>'2'</jSDL-lsf:WarningTime> <jSDL-lsf:SpoolCommand>true</jSDL-lsf:SpoolCommand> <jSDL-lsf:Checkpoint></jSDL-lsf:RuntimeParams></pre>
Interactive	-l[s p]	String that specifies an interactive job with a defined pseudo-terminal mode.
Block	-K	Boolean value that tells LSF to complete the submitted job before allowing the user to submit another job.
PreExec	-E	String that specifies a pre-exec command to run on the batch job's execution host before actually running the job. For a parallel job, the pre-exec command runs on the first host selected for the parallel job.
Checkpoint	-k	String that makes a job checkpointable and specifies the checkpoint directory.
LoginShell	-L	For UNIX jobs, string that tells LSF to initialize the execution environment using the specified login shell.

Element	bsub Option	Description
SignalJob	-s	String that specifies the signal to send when a queue-level run window closes. Use this to override the default signal that suspends jobs running in the queue.
WarningAction	-wa	String that specifies the job action prior to the job control action. Requires that you also specify the job action warning time.
WarningTime	-wt	Positive integer that specifies the amount of time prior to a job control action that the job warning action should occur.
SpoolCommand	-is	Boolean value that spools a job command file to the directory specified by JOB_SPOOL_DIR, and uses the spooled file as the command file for the job.

Unsupported JSDL and POSIX extension elements

The current version of LSF does not support the following elements:

Job structure elements

- ◆ Description

Job identity elements

- ◆ JobAnnotation

Resource elements

- ◆ FileSystem
- ◆ MountPoint
- ◆ MountSource
- ◆ DiskSpace
- ◆ FileSystemType
- ◆ OperatingSystemType
- ◆ IndividualCPUTime
- ◆ IndividualDiskSpace
- ◆ TotalCPUTime
- ◆ TotalDiskSpace

Data staging elements

- ◆ FileSystemName
- ◆ DeleteOnTermination

POSIX extension elements

- ◆ LockedMemoryLimit
- ◆ OpenDescriptorsLimit
- ◆ PipeSizeLimit
- ◆ UserName
- ◆ GroupName

Submit a job using a JSDL file

- 1 To submit a job using a JSDL file, use one of the following `bsub` command options:
 - a To submit a job that uses elements included in the LSF extension, use the `-jsdl` option.
 - b To submit a job that uses only standard JSDL elements and POSIX extensions, use the `-jsdl_strict` option. Error messages indicate invalid elements, including:
 - ◆ Elements that are not part of the JSDL specification
 - ◆ Valid JSDL elements that are not supported in this version of LSF
 - ◆ Elements that are not part of the JSDL standard and POSIX extension schemas

If you specify duplicate or conflicting job submission parameters, LSF resolves the conflict by applying the following rules:

- ◆ The parameters specified in the command line override all other parameters.
- ◆ A job script or user input for an interactive job overrides parameters specified in the JSDL file.

Collecting resource values using elim.jsdl

To support the use of JSDL files at job submission, LSF collects the following load metrics:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth (This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources, but you must enable its use by modifying the files `lsf.cluster.cluster_name` and `lsf.shared`.

Enable JSDL resource collection

- 1 In the file `lsf.cluster.cluster_name`, locate the `ResourcesMap` section.
- 2 In the file `lsf.shared`, locate the `Resource` section.
- 3 Uncomment the lines for the following resources in both files:
 - ◆ `osname`
 - ◆ `osver`

- ◆ `cpuarch`
 - ◆ `cpuspeed`
 - ◆ `bandwidth`
- 4 To propagate the changes through the LSF system, run the following commands.
- a `lsadmin reconfig`
 - b `badmin mbdrestart`

You have now configured LSF to use the `elim.jsdl` file to collect JSDL resources.

Collecting resource values using elim.jsdl



Controlling Job Execution

- ◆ [Runtime Resource Usage Limits](#) on page 509
- ◆ [Load Thresholds](#) on page 523
- ◆ [Pre-Execution and Post-Execution Commands](#) on page 529
- ◆ [Job Starters](#) on page 537
- ◆ [External Job Submission and Execution Controls](#) on page 543
- ◆ [Configuring Job Controls](#) on page 553

Runtime Resource Usage Limits

Contents

- ◆ [About Resource Usage Limits](#) on page 509
- ◆ [Specifying Resource Usage Limits](#) on page 513
- ◆ [Supported Resource Usage Limits and Syntax](#) on page 515
- ◆ [CPU Time and Run Time Normalization](#) on page 521

About Resource Usage Limits

Resource usage limits control how much resource can be consumed by running jobs. Jobs that use more than the specified amount of a resource are signalled or have their priority lowered.

Limits can be specified by the LSF administrator:

- ◆ At the queue level in `lsb.queues`
- ◆ In an application profile in `lsb.applications`
- ◆ At the job level when you submit a job

For example, by defining a high-priority short queue, you can allow short jobs to be scheduled earlier than long jobs. To prevent some users from submitting long jobs to this short queue, you can set CPU limit for the queue so that no jobs submitted from the queue can run for longer than that limit.

Limits specified at the queue level are *hard* limits, while those specified with job submission or in an application profile are *soft* limits. See `setrlimit(2)` man page for concepts of hard and soft limits.

NOTE: This chapter describes queue-level and job-level resource usage limits. Priority of limits is different if limits are also configured in an application profile. See Chapter 20, “[Working with Application Profiles](#)” for information about resource usage limits in application profiles.

Resource usage limits and resource allocation limits

Resource usage limits are not the same as *resource allocation limits*, which are enforced during job scheduling and before jobs are dispatched. You set resource allocation limits to restrict the amount of a given resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to.

See Chapter 21, “[Resource Allocation Limits](#)” for more information.

Summary of resource usage limits

Limit	Job syntax (bsub)	Syntax (lsb.queues and lsb.applications)	Format/Default Units
Core file size limit	-C <i>core_limit</i>	CORELIMIT= <i>limit</i>	<i>integer</i> KB
CPU time limit	-c <i>cpu_limit</i>	CPULIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]
Data segment size limit	-D <i>data_limit</i>	DATALIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
File size limit	-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB
Memory limit	-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
Process limit	-p <i>process_limit</i>	PROCESLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>
Run time limit	-W <i>run_limit</i>	RUNLIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]
Stack segment size limit	-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	<i>integer</i> KB
Virtual memory limit	-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	<i>integer</i> KB
Thread limit	-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Priority of resource usage limits

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
Both default and maximum limits are defined	The default is enforced
Only a maximum is defined	The maximum is enforced
No limit is specified in the queue or at job submission	No limits are enforced

Incorrect resource usage limits

Incorrect limits are ignored, and a warning message is displayed when the cluster is reconfigured or restarted. A warning message is also logged to the `mbatchd` log file when LSF is started.

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
The default limit is not correct	The default is ignored and the maximum limit is enforced
Both default and maximum limits are specified, and the maximum is not correct	The maximum is ignored and the resource has no maximum limit, only a default limit
Both default and maximum limits are not correct	The default and maximum are ignored and no limit is enforced

Resource usage limits specified at job submission must be less than the maximum specified in `lsb.queues`. The job submission is rejected if the user-specified limit is greater than the queue-level maximum, and the following message is issued:

```
Cannot exceed queue's hard limit(s). Job not submitted.
```

Enforce limits on chunk jobs

By default, resource usage limits are not enforced for chunk jobs because chunk jobs are typically too short to allow LSF to collect resource usage.

- 1 To enforce resource limits for chunk jobs, define `LSB_CHUNK_RUSAGE=Y` in `lsf.conf`. Limits may not be enforced for chunk jobs that take less than a minute to run.

Scaling the units for resource usage limits

The default unit for the following resource usage limits is KB:

- ◆ Core limit (`-C` and `CORELIMIT`)
- ◆ Memory limit (`-M` and `MEMLIMIT`)
- ◆ Stack limit (`-S` and `STACKLIMIT`)
- ◆ Swap limit (`-v` and `SWAPLIMIT`)

This default may be too small for some environments that make use of very large resource usage limits, for example, GB or TB.

`LSF_UNIT_FOR_LIMITS` in `lsf.conf` specifies larger units for the resource usage limits with default unit of KB.

The unit for the resource usage limit can be one of:

- ◆ KB (kilobytes)
- ◆ MB (megabytes)
- ◆ GB (gigabytes)
- ◆ TB (terabytes)
- ◆ PB (petabytes)
- ◆ EB (exabytes)

`LSF_UNIT_FOR_LIMITS` applies cluster-wide to limits at the job-level (`bsub`), queue-level (`lsb.queues`), and application level (`lsb.applications`).

About Resource Usage Limits

The limit unit specified by `LSF_UNIT_FOR_LIMITS` also applies to limits modified with `bmod`, and the display of resource usage limits in query commands (`bacct`, `bapp`, `bhist`, `bhosts`, `bjobs`, `bqueues`, `lslload`, and `lshosts`).

IMPORTANT: Before changing the units of your resource usage limits, you should completely drain the cluster of all workload. There should be no running, pending, or finished jobs in the system.

In a MultiCluster environment, you should configure the same unit for all clusters. After changing `LSF_UNIT_FOR_LIMITS`, you must restart your cluster.

How limit unit changes affect jobs

When `LSF_UNIT_FOR_LIMITS` is specified, the defined unit is used for the following commands. In command output, the larger unit will appear as T, G, P, or E, depending on the job rusage and the unit defined.

Command	Option/Output	Default unit
bsub/bmod	-C (core limit)	KB
	-M (memory limit)	KB
	-S (stack limit)	KB
	-v (swap limit)	KB
bjobs	rusage CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (may show MB depending on job rusage)
bqueues	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (may show MB depending on job rusage)
	loadSched, loadStop	MB
bacct	Summary rusage	KB (may show MB depending on job rusage)
bapp	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB
bhist	History of limit change by bmod	KB
	MEM, SWAP	KB (may show MB depending on job rusage)
bhosts	loadSched, loadStop	MB
lslload	mem, swp	KB (may show MB depending on job rusage)
lshosts	maxmem, maxswp	KB (may show MB depending on job rusage)

Example

A job is submitted with `bsub -M 100` and `LSF_UNIT_FOR_LIMITS=MB`; the memory limit for the job is 100 MB rather than the default 100 KB.

Specifying Resource Usage Limits

Queues can enforce resource usage limits on running jobs. LSF supports most of the limits that the underlying operating system supports. In addition, LSF also supports a few limits that the underlying operating system does not support.

Specify queue-level resource usage limits using parameters in `lsb.queues`.

Specifying queue-level resource usage limits

Limits configured in `lsb.queues` apply to all jobs submitted to the queue. Job-level resource usage limits specified at job submission override the queue definitions.

Maximum value only

Specify only a maximum value for the resource.

For example, to specify a maximum run limit, use one value for the `RUNLIMIT` parameter in `lsb.queues`:

```
RUNLIMIT = 10
```

The maximum run limit for the queue is 10 minutes. Jobs cannot run for more than 10 minutes. Jobs in the `RUN` state for longer than 10 minutes are killed by LSF.

If only one run limit is specified, jobs that are submitted with `bsub -W` with a run limit that exceeds the maximum run limit will not be allowed to run. Jobs submitted without `bsub -W` will be allowed to run but will be killed when they are in the `RUN` state for longer than the specified maximum run limit.

For example, in `lsb.queues`:

```
RUNLIMIT = 10
```

The maximum run limit for the queue is 10 minutes. Jobs cannot run for more than 10 minutes.

Default and maximum values

If you specify two limits, the first one is the default (soft) limit for jobs in the queue and the second one is the maximum (hard) limit. Both the default and the maximum limits must be positive integers. The default limit must be less than the maximum limit. The default limit is ignored if it is greater than the maximum limit.

Use the default limit to avoid having to specify resource usage limits in the `bsub` command.

For example, to specify a default and a maximum run limit, use two values for the `RUNLIMIT` parameter in `lsb.queues`:

```
RUNLIMIT = 10 15
```

- ◆ The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without `bsub -W`).
- ◆ The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with `bsub -W`). The default run limit must be less than the maximum run limit.

You can specify both default and maximum values for the following resource usage limits in `lsb.queues`:

- ◆ `CPULIMIT`
- ◆ `DATALIMIT`
- ◆ `MEMLIMIT`

Specifying Resource Usage Limits

- ◆ PROCESSLIMIT
- ◆ RUNLIMIT
- ◆ THREADLIMIT

Host specification with two limits

If default and maximum limits are specified for CPU time limits or run time limits, only one host specification is permitted. For example, the following CPU limits are correct (and have an identical effect):

```
CPULIMIT = 400/hostA 600
```

```
CPULIMIT = 400 600/hostA
```

The following CPU limit is not correct:

```
CPULIMIT = 400/hostA 600/hostB
```

The following run limits are correct (and have an identical effect):

```
RUNLIMIT = 10/hostA 15
```

```
RUNLIMIT = 10 15/hostA
```

The following run limit is not correct:

```
RUNLIMIT = 10/hostA 15/hostB
```

Default run limits for backfill scheduling

Default run limits are used for backfill scheduling of parallel jobs.

For example, in `lsb.queues`, you enter: `RUNLIMIT = 10 15`

- ◆ The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without `bsub -W`).
- ◆ The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with `bsub -W`). The default run limit cannot exceed the maximum run limit.

Automatically assigning a default run limit to all jobs in the queue means that backfill scheduling works efficiently.

For example, in `lsb.queues`, you enter:

```
RUNLIMIT = 10 15
```

The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit. The second number is the maximum run limit.

If you submit a job to the queue without the `-W` option, the default run limit is used:

```
bsub myjob
```

The job `myjob` cannot run for more than 10 minutes as specified with the default run limit.

If you submit a job to the queue with the `-W` option, the maximum run limit is used:

```
bsub -W 12 myjob
```

The job `myjob` is allowed to run on the queue because the specified run limit (12) is less than the maximum run limit for the queue (15).

```
bsub -W 20 myjob
```

The job `myjob` is rejected from the queue because the specified run limit (20) is more than the maximum run limit for the queue (15).

Specify job-level resource usage limits

1 To specify resource usage limits at the job level, use one of the following `bsub` options:

- ❖ `-C core_limit`
- ❖ `-c cpu_limit`
- ❖ `-D data_limit`
- ❖ `-F file_limit`
- ❖ `-M mem_limit`
- ❖ `-p process_limit`
- ❖ `-W run_limit`
- ❖ `-S stack_limit`
- ❖ `-T thread_limit`
- ❖ `-v swap_limit`

Job-level resource usage limits specified at job submission override the queue definitions.

Supported Resource Usage Limits and Syntax

Core file size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-C core_limit</code>	<code>CORELIMIT=limit</code>	<i>integer</i> KB

Sets a per-process (soft) core file size limit for each process that belongs to this batch job.

By default, the limit is specified in KB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

On some systems, no core file is produced if the image for the process is larger than the core limit. On other systems only the first *core_limit* KB of the image are dumped. The default is no soft limit.

CPU time limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-c cpu_limit</code>	<code>CPULIMIT=[default] maximum</code>	<i>[hours:]minutes[/host_name /host_model]</i>

Sets the soft CPU time limit to *cpu_limit* for this batch job. The default is no limit. This option is useful for avoiding runaway jobs that use up too many resources. LSF keeps track of the CPU time used by all processes of the job.

When the job accumulates the specified amount of CPU time, a SIGXCPU signal is sent to all processes belonging to the job. If the job has no signal handler for SIGXCPU, the job is killed immediately. If the SIGXCPU signal is handled, blocked, or ignored by the application, then after the grace period expires, LSF sends SIGINT, SIGTERM, and SIGKILL to the job to kill it.

You can define whether the CPU limit is a per-process limit enforced by the OS or a per-job limit enforced by LSF with `LSB_JOB_CPULIMIT` in `lsf.conf`.

Jobs submitted to a chunk job queue are not chunked if the CPU limit is greater than 30 minutes.

Format

cpu_limit is in the form `[hour:]minute`, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

Normalized CPU time

The CPU time limit is *normalized* according to the CPU factor of the submission host and execution host. The CPU limit is scaled so that the job does approximately the same amount of processing for a given CPU limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the CPU time limit is multiplied by $2/3$ because the execution host can do the same amount of work as the submission host in $2/3$ of the time.

If the optional host name or host model is not given, the CPU limit is scaled based on the `DEFAULT_HOST_SPEC` specified in the `lsb.params` file. (If `DEFAULT_HOST_SPEC` is not defined, the fastest batch host in the cluster is used as the default.) If host or host model is given, its CPU scaling factor is used to adjust the actual CPU time limit at the execution host.

The following example specifies that `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -c 10/DEC3000 myjob
```

See [CPU Time and Run Time Normalization](#) on page 521 for more information.

Data segment size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-D data_limit</code>	<code>DATALIMIT=[default] maximum</code>	<i>integer</i> KB

Sets a per-process (soft) data segment size limit in KB for each process that belongs to this batch job (see `getrlimit(2)`).

This option affects calls to `sbrk()` and `brk()`. An `sbrk()` or `malloc()` call to extend the data segment beyond the data limit returns an error.

NOTE: Linux does not use `sbrk()` and `brk()` within its `calloc()` and `malloc()`. Instead, it uses `mmap()` to create memory. `DATALIMIT` cannot be enforced on Linux applications that call `sbrk()` and `malloc()`.

The default is no soft limit.

File size limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB

Sets a per-process (soft) file size limit in KB for each process that belongs to this batch job. If a process of this job attempts to write to a file such that the file size would increase beyond the file limit, the kernel sends that process a SIGXFSZ signal. This condition normally terminates the process, but may be caught. The default is no soft limit.

Memory limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB

Sets a per-process physical memory limit for all of the processes belonging to a job. By default, the limit is specified in KB. Use LSF_UNIT_FOR_LIMITS in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

If LSB_MEMLIMIT_ENFORCE=Y or LSB_JOB_MEMLIMIT=Y are set in `lsf.conf`, LSF kills the job when it exceeds the memory limit. Otherwise, LSF passes the memory limit to the operating system. Some operating systems apply the memory limit to each process, and some do not enforce the memory limit at all.

LSF memory limit enforcement

To enable LSF memory limit enforcement, set LSB_MEMLIMIT_ENFORCE in `lsf.conf` to `y`. LSF memory limit enforcement explicitly sends a signal to kill a running process once it has allocated memory past *mem_limit*.

You can also enable LSF memory limit enforcement by setting LSB_JOB_MEMLIMIT in `lsf.conf` to `y`. The difference between LSB_JOB_MEMLIMIT set to `y` and LSB_MEMLIMIT_ENFORCE set to `y` is that with LSB_JOB_MEMLIMIT, only the per-job memory limit enforced by LSF is enabled. The per-process memory limit enforced by the OS is disabled. With LSB_MEMLIMIT_ENFORCE set to `y`, both the per-job memory limit enforced by LSF and the per-process memory limit enforced by the OS are enabled.

LSB_JOB_MEMLIMIT disables per-process memory limit enforced by the OS and enables per-job memory limit enforced by LSF. When the total memory allocated to all processes in the job exceeds the memory limit, LSF sends the following signals to kill the job: SIGINT first, then SIGTERM, then SIGKILL.

On UNIX, the time interval between SIGINT, SIGKILL, SIGTERM can be configured with the parameter JOB_TERMINATE_INTERVAL in `lsb.params`.

OS memory limit enforcement

OS enforcement usually allows the process to eventually run to completion. LSF passes *mem_limit* to the OS, which uses it as a guide for the system scheduler and memory allocator. The system may allocate more memory to a process if there is a surplus. When memory is low, the system takes memory from and lowers the scheduling priority (re-nice) of a process that has exceeded its declared *mem_limit*.

OS memory limit enforcement is only available on systems that support RLIMIT_RSS for `setrlimit()`.

The following operating systems do not support the memory limit at the OS level:

- ◆ Microsoft Windows
- ◆ Sun Solaris 2.x

Process limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-p <i>process_limit</i>	PROCESSLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Sets the limit of the number of processes to *process_limit* for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

Limits the number of concurrent processes that can be part of a job.

If a default process limit is specified, jobs submitted to the queue without a job-level process limit are killed when the default process limit is reached.

If you specify only one limit, it is the maximum, or hard, process limit. If you specify two limits, the first one is the default, or soft, process limit, and the second one is the maximum process limit.

Run time limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-W <i>run_limit</i>	RUNLIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]

A run time limit is the maximum amount of time a job can run before it is terminated. It sets the run time limit of a job. The default is no limit. If the accumulated time the job has spent in the RUN state exceeds this limit, the job is sent a USR2 signal. If the job does not terminate within 10 minutes after being sent this signal, it is killed.

With deadline constraint scheduling configured, a run limit also specifies the amount of time a job is expected to take, and the minimum amount of time that must be available before a job can be started.

Jobs submitted to a chunk job queue are not chunked if the run limit is greater than 30 minutes.

Format

run_limit is in the form [*hour*:]*minute*, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

Normalized run time

The run time limit is *normalized* according to the CPU factor of the submission host and execution host. The run limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the run limit is multiplied by 2/3 because the execution host can do the same amount of work as the submission host in 2/3 of the time.

If the optional host name or host model is not given, the run limit is scaled based on the DEFAULT_HOST_SPEC specified in the `lsb.params` file. (If DEFAULT_HOST_SPEC is not defined, the fastest batch host in the cluster is used as the default.) If host or host model is given, its CPU scaling factor is used to adjust the actual run limit at the execution host.

The following example specifies that `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -W 10/DEC3000 myjob
```

If ABS_RUNLIMIT=Y is defined in `lsb.params`, the run time limit is *not* normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit.

See [CPU Time and Run Time Normalization](#) on page 521 for more information.

Platform MultiCluster

For MultiCluster jobs, if no other CPU time normalization host is defined and information about the submission host is not available, LSF uses the host with the largest CPU factor (the fastest host in the cluster). The ABS_RUNLIMIT parameter in `lsb.params` is not supported in either MultiCluster model; run time limit is normalized by the CPU factor of the execution host.

Thread limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Sets the limit of the number of concurrent threads to *thread_limit* for the whole job. The default is no limit.

Exceeding the limit causes the job to terminate. The system sends the following signals in sequence to all processes belongs to the job: SIGINT, SIGTERM, and SIGKILL.

If a default thread limit is specified, jobs submitted to the queue without a job-level thread limit are killed when the default thread limit is reached.

If you specify only one limit, it is the maximum, or hard, thread limit. If you specify two limits, the first one is the default, or soft, thread limit, and the second one is the maximum thread limit.

Stack segment size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	<i>integer</i> KB

Sets a per-process (soft) stack segment size limit for all of the processes belonging to a job.

By default, the limit is specified in KB. Use LSF_UNIT_FOR_LIMITS in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

An `sbrk()` call to extend the stack segment beyond the stack limit causes the process to be terminated. The default is no soft limit.

Virtual memory (swap) limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	<i>integer</i> KB

Sets a total process virtual memory limit for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

By default, the limit is specified in KB. Use LSF_UNIT_FOR_LIMITS in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

This limit applies to the whole job, no matter how many processes the job may contain.

Examples

Queue-level limits

```
CPULIMIT = 20/hostA 15
```

The first number is the default CPU limit. The second number is the maximum CPU limit.

However, the default CPU limit is ignored because it is a higher value than the maximum CPU limit.

```
CPULIMIT = 10/hostA
```

In this example, the lack of a second number specifies that there is no default CPU limit. The specified number is considered as the default and maximum CPU limit.

```
RUNLIMIT = 10/hostA 15
```

The first number is the default run limit. The second number is the maximum run limit.

The first number specifies that the default run limit is to be used for jobs that are submitted without a specified run limit (without the `-W` option of `bsub`).

```
RUNLIMIT = 10/hostA
```

No default run limit is specified. The specified number is considered as the default and maximum run limit.

```
THREADLIMIT=6
```

No default thread limit is specified. The value 6 is the default and maximum thread limit.

```
THREADLIMIT=6 8
```

The first value (6) is the default thread limit. The second value (8) is the maximum thread limit.

Job-level limits

```
bsub -M 5000 myjob
```

Submits `myjob` with a memory limit of 5000 KB.

```
bsub -W 14 myjob
```

`myjob` is expected to run for 14 minutes. If the run limit specified with `bsub -W` exceeds the value for the queue, the job will be rejected.


```
bsub -T 4 myjob
```

Submits `myjob` with a maximum number of concurrent threads of 4.

CPU Time and Run Time Normalization

To set the CPU time limit and run time limit for jobs in a platform-independent way, LSF scales the limits by the CPU factor of the hosts involved. When a job is dispatched to a host for execution, the limits are then normalized according to the CPU factor of the execution host.

Whenever a normalized CPU time or run time is given, the actual time on the execution host is the specified time multiplied by the CPU factor of the normalization host then divided by the CPU factor of the execution host.

If `ABS_RUNLIMIT=Y` is defined in `lsb.params` or in `lsb.applications` for the application associated with your job, the run time limit and run time estimate are not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run time limit or a run time estimate.

Normalization host

If no host or host model is given with the CPU time or run time, LSF uses the default CPU time normalization host defined at the queue level (`DEFAULT_HOST_SPEC` in `lsb.queues`) if it has been configured, otherwise uses the default CPU time normalization host defined at the cluster level (`DEFAULT_HOST_SPEC` in `lsb.params`) if it has been configured, otherwise uses the submission host.

Example

```
CPULIMIT=10/hostA
```

If `hostA` has a CPU factor of 2, and `hostB` has a CPU factor of 1 (`hostB` is slower than `hostA`), this specifies an actual time limit of 10 minutes on `hostA`, or on any other host that has a CPU factor of 2. However, if `hostB` is the execution host, the actual time limit on `hostB` is 20 minutes ($10 * 2 / 1$).

Normalization hosts for default CPU and run time limits

The first valid CPU factor encountered is used for both CPU limit and run time limit. To be valid, a host specification must be a valid host name that is a member of the LSF cluster. The CPU factor is used even if the specified limit is not valid.

If the CPU and run limit have different host specifications, the CPU limit host specification is enforced.

If no host or host model is given with the CPU or run time limits, LSF determines the default normalization host according to the following priority:

- 1 `DEFAULT_HOST_SPEC` is configured in `lsb.queues`
- 2 `DEFAULT_HOST_SPEC` is configured in `lsb.params`
- 3 If `DEFAULT_HOST_SPEC` is not configured in `lsb.queues` or `lsb.params`, host with the largest CPU factor is used.

CPU time display (`bacct`, `bhist`, `bqueues`)

Normalized CPU time is displayed in the output of `bqueues`. CPU time is *not* normalized in the output if `bacct` and `bhist`.

Load Thresholds

Contents

- ◆ [Automatic Job Suspension](#) on page 523
- ◆ [Suspending Conditions](#) on page 524

Automatic Job Suspension

Jobs running under LSF can be suspended based on the load conditions on the execution hosts. Each host and each queue can be configured with a set of suspending conditions. If the load conditions on an execution host exceed either the corresponding host or queue suspending conditions, one or more jobs running on that host will be suspended to reduce the load.

When LSF suspends a job, it invokes the SUSPEND action. The default SUSPEND action is to send the signal SIGSTOP.

By default, jobs are resumed when load levels fall below the suspending conditions. Each host and queue can be configured so that suspended checkpointable or rerunnable jobs are automatically migrated to another host instead.

If no suspending threshold is configured for a load index, LSF does not check the value of that load index when deciding whether to suspend jobs.

Suspending thresholds can also be used to enforce inter-queue priorities. For example, if you configure a low-priority queue with an `r1m` (1 minute CPU run queue length) scheduling threshold of 0.25 and an `r1m` suspending threshold of 1.75, this queue starts one job when the machine is idle. If the job is CPU intensive, it increases the run queue length from 0.25 to roughly 1.25. A high-priority queue configured with a scheduling threshold of 1.5 and an unlimited suspending threshold will send a second job to the same host, increasing the run queue to 2.25. This exceeds the suspending threshold for the low priority job, so it is stopped. The run queue length stays above 0.25 until the high priority job exits. After the high priority job exits the run queue index drops back to the idle level, so the low priority job is resumed.

Suspending Conditions

When jobs are running on a host, LSF periodically checks the load levels on that host. If any load index exceeds the corresponding per-host or per-queue suspending threshold for a job, LSF suspends the job. The job remains suspended until the load levels satisfy the scheduling thresholds.

At regular intervals, LSF gets the load levels for that host. The period is defined by the `SBD_SLEEP_TIME` parameter in the `lsb.params` file. Then, for each job running on the host, LSF compares the load levels against the host suspending conditions and the queue suspending conditions. If any suspending condition at either the corresponding host or queue level is satisfied as a result of increased load, the job is suspended. A job is only suspended if the load levels are too high for that particular job's suspending thresholds.

There is a time delay between when LSF suspends a job and when the changes to host load are seen by the LIM. To allow time for load changes to take effect, LSF suspends no more than one job at a time on each host.

Jobs from the lowest priority queue are checked first. If two jobs are running on a host and the host is too busy, the lower priority job is suspended and the higher priority job is allowed to continue. If the load levels are still too high on the next turn, the higher priority job is also suspended.

If a job is suspended because of its own load, the load drops as soon as the job is suspended. When the load goes back within the thresholds, the job is resumed until it causes itself to be suspended again.

Exceptions

In some special cases, LSF does not automatically suspend jobs because of load levels. LSF does not suspend a job:

- ◆ Forced to run with `brun -f`.
- ◆ If it is the only job running on a host, unless the host is being used interactively. When only one job is running on a host, it is not suspended for any reason except that the host is not interactively idle (the `it` interactive idle time load index is less than one minute). This means that once a job is started on a host, at least one job continues to run unless there is an interactive user on the host. Once the job is suspended, it is not resumed until all the scheduling conditions are met, so it should not interfere with the interactive user.
- ◆ Because of the paging rate, unless the host is being used interactively. When a host has interactive users, LSF suspends jobs with high paging rates, to improve the response time on the host for interactive users. When a host is idle, the `pg` (paging rate) load index is ignored. The `PG_SUSP_IT` parameter in `lsb.params` controls this behaviour. If the host has been idle for more than `PG_SUSP_IT` minutes, the `pg` load index is not checked against the suspending threshold.

Suspending Conditions

LSF provides different alternatives for configuring suspending conditions. Suspending conditions are configured at the host level as load thresholds, whereas suspending conditions are configured at the queue level as either load thresholds, or by using the `STOP_COND` parameter in the `lsb.queues` file, or both.

The load indices most commonly used for suspending conditions are the CPU run queue lengths (`r15s`, `r1m`, and `r15m`), paging rate (`pg`), and idle time (`it`). The (`swp`) and (`tmp`) indices are also considered for suspending jobs.

To give priority to interactive users, set the suspending threshold on the `it` (idle time) load index to a non-zero value. Jobs are stopped when any user is active, and resumed when the host has been idle for the time given in the `it` scheduling condition.

To tune the suspending threshold for paging rate, it is desirable to know the behaviour of your application. On an otherwise idle machine, check the paging rate using `lsload`, and then start your application. Watch the paging rate as the application runs. By subtracting the active paging rate from the idle paging rate, you get a number for the paging rate of your application. The suspending threshold should allow at least 1.5 times that amount. A job can be scheduled at any paging rate up to the scheduling threshold, so the suspending threshold should be at least the scheduling threshold plus 1.5 times the application paging rate. This prevents the system from scheduling a job and then immediately suspending it because of its own paging.

The effective CPU run queue length condition should be configured like the paging rate. For CPU-intensive sequential jobs, the effective run queue length indices increase by approximately one for each job. For jobs that use more than one process, you should make some test runs to determine your job's effect on the run queue length indices. Again, the suspending threshold should be equal to at least the scheduling threshold plus 1.5 times the load for one job.

Configuring load thresholds at queue level

The queue definition (`lsb.queues`) can contain thresholds for 0 or more of the load indices. Any load index that does not have a configured threshold has no effect on job scheduling.

Syntax

Each load index is configured on a separate line with the format:

```
load_index = loadSched/loadStop
```

Specify the name of the load index, for example `r1m` for the 1-minute CPU run queue length or `pg` for the paging rate. `loadSched` is the scheduling threshold for this load index. `loadStop` is the suspending threshold. The `loadSched` condition must be satisfied by a host before a job is dispatched to it and also before a job suspended on a host can be resumed. If the `loadStop` condition is satisfied, a job is suspended.

The `loadSched` and `loadStop` thresholds permit the specification of conditions using simple AND/OR logic. For example, the specification:

```
MEM=100/10
SWAP=200/30
```

translates into a `loadSched` condition of `mem>=100 && swap>=200` and a `loadStop` condition of `mem < 10 || swap < 30`.

Theory

- ◆ The `r15s`, `r1m`, and `r15m` CPU run queue length conditions are compared to the effective queue length as reported by `lsload -E`, which is normalised for multiprocessor hosts. Thresholds for these parameters should be set at appropriate levels for single processor hosts.

Suspending Conditions

- ◆ Configure load thresholds consistently across queues. If a low priority queue has higher suspension thresholds than a high priority queue, then jobs in the higher priority queue will be suspended before jobs in the low priority queue.

Configuring load thresholds at host level

A shared resource cannot be used as a load threshold in the `Hosts` section of the `lsf.cluster.cluster_name` file.

Configuring suspending conditions at queue level

The condition for suspending a job can be specified using the queue-level `STOP_COND` parameter. It is defined by a resource requirement string. Only the `select` section of the resource requirement string is considered when stopping a job. All other sections are ignored.

This parameter provides similar but more flexible functionality for `loadStop`.

If `loadStop` thresholds have been specified, then a job will be suspended if either the `STOP_COND` is `TRUE` or the `loadStop` thresholds are exceeded.

Example

This queue will suspend a job based on the idle time for desktop machines and based on availability of swap and memory on compute servers. Assume `cs` is a Boolean resource defined in the `lsf.shared` file and configured in the `lsf.cluster.cluster_name` file to indicate that a host is a compute server:

```
Begin Queue
.  
STOP_COND= select[(!cs && it < 5) || (cs && mem < 15 && swap < 50)]  
.  
End Queue
```

Viewing host-level and queue-level suspending conditions

The suspending conditions are displayed by the `bhosts -l` and `bqueues -l` commands.

Viewing job-level suspending conditions

The thresholds that apply to a particular job are the more restrictive of the host and queue thresholds, and are displayed by the `bjobs -l` command.

Viewing suspend reason

The `bjobs -lp` command shows the load threshold that caused LSF to suspend a job, together with the scheduling parameters.

The use of `STOP_COND` affects the suspending reasons as displayed by the `bjobs` command. If `STOP_COND` is specified in the queue and the `loadStop` thresholds are not specified, the suspending reasons for each individual load index will not be displayed.

Resuming suspended jobs

Jobs are suspended to prevent overloading hosts, to prevent batch jobs from interfering with interactive use, or to allow a more urgent job to run. When the host is no longer overloaded, suspended jobs should continue running.

When LSF automatically resumes a job, it invokes the RESUME action. The default action for RESUME is to send the signal SIGCONT.

If there are any suspended jobs on a host, LSF checks the load levels in each dispatch turn.

If the load levels are within the scheduling thresholds for the queue and the host, and all the resume conditions for the queue (RESUME_COND in `lsb.queues`) are satisfied, the job is resumed.

If RESUME_COND is not defined, then the `loadSched` thresholds are used to control resuming of jobs: all the `loadSched` thresholds must be satisfied for the job to be resumed. The `loadSched` thresholds are ignored if RESUME_COND is defined.

Jobs from higher priority queues are checked first. To prevent overloading the host again, only one job is resumed in each dispatch turn.

Specify resume condition

- 1 Use RESUME_COND in `lsb.queues` to specify the condition that must be satisfied on a host if a suspended job is to be resumed.
Only the `select` section of the resource requirement string is considered when resuming a job. All other sections are ignored.

Viewing resume thresholds

The `bjobs -l` command displays the scheduling thresholds that control when a job is resumed.

Suspending Conditions

Pre-Execution and Post-Execution Commands

Jobs can be submitted with optional pre- and post-execution commands. A pre- or post-execution command is an arbitrary command to run before the job starts or after the job finishes. Pre- and post-execution commands are executed in a separate environment from the job.

Contents

- ◆ [About Pre-Execution and Post-Execution Commands](#) on page 529
- ◆ [Configuring Pre- and Post-Execution Commands](#) on page 531

About Pre-Execution and Post-Execution Commands

Each batch job can be submitted with optional pre- and post-execution commands. Pre- and post-execution commands can be any executable command lines to be run before a job is started or after a job finishes.

Some batch jobs require resources that LSF does not directly support. For example, appropriate pre- and/or post-execution commands can be used to handle various situations:

- ◆ Reserving devices like tape drives
- ◆ Creating and deleting scratch directories for a job
- ◆ Customized scheduling
- ◆ Checking availability of software licenses
- ◆ Assigning jobs to run on specific processors on SMP machines
- ◆ Transferring data files needed for job processing
- ◆ Modifying system configuration files before and after running a job

By default, the pre- and post-execution commands are run under the same user ID, environment, and home and working directories as the batch job. If the command is not in your normal execution path, the full path name of the command must be specified.

For parallel jobs, the command is run on the first selected host.

Pre-execution commands

Pre-execution commands support job starting decisions which cannot be configured directly in LSF. LSF supports job-level, queue-level, and application-level (`lsb.applications`) pre-execution.

The pre-execution command returns information to LSF using its exit status. When a pre-execution command is specified, the job is held in the queue until the specified pre-execution command returns exit status zero (0).

If the pre-execution command exits with non-zero status, the batch job is not dispatched. The job goes back to the PEND state, and LSF tries to dispatch another job to that host. While the job is pending, other jobs can proceed ahead of the waiting job. The next time LSF tries to dispatch jobs this process is repeated.

If the pre-execution command exits with a value of 99, the job will not go back to the PEND state, it will exit. This gives you flexibility to abort the job if the pre-execution command fails.

LSF assumes that the pre-execution command runs without side effects. For example, if the pre-execution command reserves a software license or other resource, you must not reserve the same resource more than once for the same batch job.

Post-execution commands

If a post-execution command is specified, then the command is run after the job is finished regardless of the exit state of the job.

Post-execution commands are typically used to clean up some state left by the pre-execution and the job execution. LSF supports job-level, queue-level, and application-level (`lsb.applications`) post-execution.

Job-level commands

The `bsub -E` option specifies an arbitrary command to run before starting the batch job. When LSF finds a suitable host on which to run a job, the pre-execution command is executed on that host. If the pre-execution command runs successfully, the batch job is started.

The `bsub -Ep` option specifies job-level post-execution commands to run on the execution host after the job finishes.

Queue-level and application-level commands

In some situations (for example, license checking), it is better to specify a queue-level or application-level pre-execution command instead of requiring every job be submitted with the `-E` option of `bsub`.

Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (`bsub -E`) override application-level pre-execution commands.

Application level pre-execution commands run on the execution host before the job associated with the application profile is dispatched on an execution host.

When a job finishes, the application-level post-execution commands run, followed by queue-level post-execution commands if any.

Application-level post-execution commands run on the execution host after the job associated with the application profile has finished running on the execution host. They also run if the PRE_EXEC command exits with a 0 exit status, but the job execution environment failed to be set up.

Post-execution job states

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

By default, the DONE or EXIT job states do not indicate whether post-processing is complete, so jobs that depend on processing may start prematurely. Use the `post_done` and `post_err` keywords on the `bsub -w` command to specify job dependency conditions for job post-processing. The corresponding job states POST_DONE and POST_ERR indicate the state of the post-processing.

The `bhist` command displays the POST_DONE and POST_ERR states. The resource usage of post-processing is not included in the job resource usage.

After the job completes, you cannot perform any job control on the post-processing. Post-processing exit codes are not reported to LSF.

Configuring Pre- and Post-Execution Commands

Pre-execution commands can be configured at the job level, in queues, or in application profiles.

Post-execution commands can be configured at the job level, in queues or in application profiles.

Job-level commands

Job-level pre-execution and post-execution commands require no configuration. Use the `bsub -E` option to specify an arbitrary command to run before the job starts. Use the `bsub -Ep` option to specify an arbitrary command to run after the job finishes running.

Example

The following example shows a batch job that requires a tape drive. The user program `tapeCheck` exits with status zero if the specified tape drive is ready:

```
bsub -E "/usr/share/bin/tapeCheck /dev/rmt01" myJob
```

Queue-level commands

Use the PRE_EXEC and POST_EXEC keywords in the queue definition (`lsb.queues`) to specify pre- and post-execution commands.

The following points should be considered when setting up pre- and post-execution commands at the queue level:

- ◆ If the pre-execution command exits with a non-zero exit code, then it is considered to have failed and the job is requeued to the head of the queue. This feature can be used to implement customized scheduling by having the pre-execution command fail if conditions for dispatching the job are not met.
- ◆ Other environment variables set for the job are also set for the pre- and post-execution commands.

- ◆ When a job is dispatched from a queue which has a post-execution command, LSF will remember the post-execution command defined for the queue from which the job is dispatched. If the job is later switched to another queue or the post-execution command of the queue is changed, LSF will still run the original post-execution command for this job.
- ◆ When the post-execution command is run, the environment variable, `LSB_JOBEXIT_STAT`, is set to the exit status of the job. See the man page for the `wait(2)` command for the format of this exit status.
- ◆ The post-execution command is also run if a job is requeued because the job's execution environment fails to be set up, or if the job exits with one of the queue's `REQUEUE_EXIT_VALUES`. The `LSB_JOBPEND` environment variable is set if the job is requeued. If the job's execution environment could not be set up, `LSB_JOBEXIT_STAT` is set to 0.
- ◆ Running of post-execution commands upon restart of a rerunnable job may not always be desirable; for example, if the post-exec removes certain files, or does other cleanup that should only happen if the job finishes successfully. Use `LSB_DISABLE_RERUN_POST_EXEC=Y` in `lsf.conf` to prevent the post-exec from running when a job is rerun.
- ◆ If both queue and job-level pre-execution commands are specified, the job-level pre-execution is run after the queue-level pre-execution command.
- ◆ If both application-level and job-level post-execution commands are specified, job level post-execution overrides application-level post-execution commands. Queue-level post-execution commands run after application-level post-execution and job-level post-execution commands

Example

The following queue specifies the pre-execution command `/usr/share/lsf/pri_prexec` and the post-execution command `/usr/share/lsf/pri_postexec`.

```
Begin Queue
QUEUE_NAME      = priority
PRIORITY        = 43
NICE             = 10
PRE_EXEC        = /usr/share/lsf/pri_prexec
POST_EXEC       = /usr/share/lsf/pri_postexec
End Queue
```

See the `lsb.queues` template file for additional queue examples.

Application-level commands

Use the `PRE_EXEC` and `POST_EXEC` keywords in the application profile definition (`lsb.applications`) to specify pre- and post-execution commands.

The following points should be considered when setting up pre- and post-execution commands at the application level:

- ◆ When a job finishes, the application-level post-execution commands run, followed by queue-level post-execution commands if any.
- ◆ Environment variables set for the job are also set for the pre- and post-execution commands.

- ◆ Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (`bsub -E`) override application-level pre-execution commands.
- ◆ When a job is submitted to an application profile that has a pre-execution command, the system will remember the post-execution command defined for the application profile from which the job is dispatched. If the job is later moved to another application profile or the post-execution command of the application profile is changed, the original post-execution command will be run.
- ◆ When the post-execution command is run, the environment variable `LSB_JOBEXIT_STAT` is set to the exit status of the job. Refer to the man page for `wait(2)` for the format of this exit status.
- ◆ The post-execution command is also run if a job is requeued because the job's execution environment fails to be set up or if the job exits with one of the application profile's `REQUEUE_EXIT_VALUES`. The environment variable `LSB_JOBPEND` is set if the job is requeued. If the job's execution environment could not be set up, `LSB_JOBEXIT_STAT` is set to 0 (zero).
- ◆ If the pre-execution command exits with a non-zero exit code, it is considered to have failed, and the job is requeued to the head of the queue. Use this feature to implement customized scheduling by having the pre-execution command fail if conditions for dispatching the job are not met.

Example

```
Begin Application
NAME          = catia
DESCRIPTION   = CATIA V5
CPULIMIT      = 24:0/hostA      # 24 hours of host hostA
FILELIMIT     = 20000
DATALIMIT     = 20000          # jobs data segment limit
CORELIMIT     = 20000
PROCLIMIT     = 5              # job processor limit
PRE_EXEC      = /usr/share/lsf/catia_prexec
POST_EXEC     = /usr/share/lsf/catia_postexec
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

See the `lsb.applications` template file for additional application profile examples.

Pre- and post-execution on UNIX and Linux

The entire contents of the configuration line of the pre- and post-execution commands are run under `/bin/sh -c`, so shell features can be used in the command.

For example, the following is valid:

```
PRE_EXEC = /usr/share/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC = /usr/share/lsf/misc/testq_post | grep -v "Hey!"
```

The pre- and post-execution commands are run in `/tmp`.

Standard input and standard output and error are set to `/dev/null`. The output from the pre- and post-execution commands can be explicitly redirected to a file for debugging purposes.

The `PATH` environment variable is set to:

```
PATH='/bin /usr/bin /sbin /usr/sbin'
```

Pre- and post-execution on Windows

The pre- and post-execution commands are run under `cmd.exe /c`.

NOTE: For pre- and post-execution commands that execute on a Windows Server 2003, x64 Edition platform, users must have “Read” and “Execute” privileges for `cmd.exe`.

Standard input and standard output and error are set to `NULL`. The output from the pre- and post-execution commands can be explicitly redirected to a file for debugging purposes.

Setting a pre- and post-execution user ID

By default, both the pre- and post-execution commands are run as the job submission user. Use the `LSB_PRE_POST_EXEC_USER` parameter in `lsf.sudoers` to specify a different user ID for queue-level and application-level pre- and post-execution commands.

Example

If the pre- or post-execution commands perform privileged operations that require root permission, specify:

```
LSB_PRE_POST_EXEC_USER=root
```

See the *Platform LSF Configuration Reference* for information about the `lsf.sudoers` file.

Including job post-execution in job finish status reporting

By default, LSF releases resources for a job as soon as the job is finished and when `sbatchd` reports job finish status (`DONE` or `EXIT`) to `mbatchd`. Post-execution processing is not considered part of job processing. This makes it possible for a new job to be started before post-execution processing for a previous job is complete.

There are situations where you do not want the first job's post-execution affecting the second job's execution. Or the execution of a second job might crucially depend on the completion of post-execution of the previous job.

In other cases, you may want to include job post-execution in job accounting processes, or if the post-execution is CPU intensive, you might not want a second job running at the same time. Finally, system configuration required by the original job may be changed or removed by a new job, which could prevent the first job from finishing normally.

To enable all associated processing to complete before LSF reports job finish status, configure `JOB_INCLUDE_POSTPROC=Y` in an application profile in `lsb.applications` or cluster wide in `lsb.params`.

When `JOB_INCLUDE_POSTPROC` is set:

- ◆ `sbatchd` sends both job finish status (`DONE` or `EXIT`) and post-execution status (`POST_DONE` or `POST_ERR`) to `mbatchd` at the same time

- ◆ The job remains in RUN state and holds its job slot until the job post-execution processing has finished
- ◆ Jobs can now depend on the completion of post-execution processing
- ◆ `bjobs`, `bhist`, and `bacct` will show the same time for both job finish and post-execution finish
- ◆ Job requeue will happen after post-execution processing, not when the job finishes

For job history and job accounting, the job CPU time and run time will also include the post-execution CPU time and run time.

Limitations and side-effects

Job query commands (`bjobs`, `bhist`) show that the job remains in RUN state until the post-execution processing is finished, even though the job itself has finished. Job control commands (`bstop`, `bkill`, `bresume`) will have no effect.

Rerunnable jobs may rerun after they have actually finished because the host became unavailable before post-execution processing finished, but the `mbatchd` considers the job still in RUN state.

Job preemption is delayed until post-execution processing is finished.

Post-execution on SGI cpusets

Post-execution processing on SGI cpusets behave differently from previous releases. If `JOB_INCLUDE_POSTPROC=Y` is specified in `lsb.applications` or cluster wide in `lsb.params`, post-execution processing is not attached to the job cpuset, and Platform LSF does not release the cpuset until post-execution processing has finished.

Preventing job overlap on hosts

You can use `JOB_INCLUDE_POSTPROC` to ensure that there is no execution overlap among running jobs. For example, you may have pre-execution processing to create a user execution environment at the desktop (mount a disc for the user, create `rlogin` permissions, etc.) Then you configure post-execution processing to clean up the user execution environment set by the `pre-exec`.

If the post-execution for one job is still running when a second job is dispatched, pre-execution processing that sets up the user environment for the next job may not be able to run correctly because the previous job's environment has not yet been cleaned up by its `post-exec`.

You should configure jobs to run exclusively to prevent the actual jobs from not overlapping, but in this case, you also need to configure post-execution to be included in job finish status reporting.

Setting a post-execution timeout

Configure `JOB_POSTPROC_TIMEOUT` in an application profile in `lsb.applications` or cluster wide in `lsb.params` to control how long post-execution processing is allowed to run.

`JOB_POSTPROC_TIMEOUT` specifies a timeout in minutes for job post-execution processing. If post-execution processing takes longer than the timeout, `sbatchd` reports the post-execution has failed (`POST_ERR` status), and kills the process group of the job's post-execution processes.

The specified timeout must be greater than zero.

Configuring Pre- and Post-Execution Commands

If `JOB_INCLUDE_POSTPROC` is enabled in the application profile or cluster wide in `lsb.params`, and `sbatchd` kills the post-execution processes because the timeout has been reached, the CPU time of the post-execution processing is set to 0, and the job CPU time will not include the CPU time of the post-execution processing.

Job Starters

A *job starter* is a specified shell script or executable program that sets up the environment for a job and then runs the job. The job starter and the job share the same environment. This chapter discusses two ways of running job starters in LSF and how to set up and use them.

Contents

- ◆ [About Job Starters](#) on page 537
- ◆ [Command-Level Job Starters](#) on page 538
- ◆ [Queue-Level Job Starters](#) on page 540
- ◆ [Controlling Execution Environment Using Job Starters](#) on page 541

About Job Starters

Some jobs have to run in a particular environment, or require some type of setup to be performed before they run. In a shell environment, job setup is often written into a wrapper shell script file that itself contains a call to start the desired job.

A job starter is a specified wrapper script or executable program that typically performs environment setup for the job, then calls the job itself, which inherits the execution environment created by the job starter. LSF controls the job starter process, rather than the job. One typical use of a job starter is to customize LSF for use with specific application environments, such as Alias Renderer or IBM Rational ClearCase.

Two ways to run job starters

You run job starters two ways in LSF. You can accomplish similar things with either job starter, but their functional details are slightly different.

Command-level

Are user-defined. They run interactive jobs submitted using `lsrun`, `lsgrun`, or `ch`. Command-level job starters have no effect on batch jobs, including interactive batch jobs run with `bsub -I`.

Use the `LSF_JOB_STARTER` environment variable to specify a job starter for interactive jobs. See [Controlling Execution Environment Using Job Starters](#) on page 541 for detailed information.

Queue-level

Defined by the LSF administrator, and run batch jobs submitted to a queue defined with the `JOB_STARTER` parameter set. Use `bsub` to submit jobs to queues with job-level job starters.

A queue-level job starter is configured in the queue definition in `lsb.queues`. See [Queue-Level Job Starters](#) on page 540 for detailed information.

Pre-execution commands are not job starters

A job starter differs from a pre-execution command. A pre-execution command must run successfully and exit before the LSF job starts. It can signal LSF to dispatch the job, but because the pre-execution command is an unrelated process, it does not control the job or affect the execution environment of the job. A job starter, however, is the process that LSF controls. It is responsible for invoking LSF and controls the execution environment of the job.

See Chapter 35, “[Pre-Execution and Post-Execution Commands](#)” for more information.

Examples

The following are some examples of job starters:

- ◆ In UNIX, a job starter defined as `/bin/ksh -c` causes jobs to be run under a Korn shell environment.
- ◆ In Windows, a job starter defined as `C:\cmd.exe /C` causes jobs to be run under a DOS shell environment.

NOTE: For job starters that execute on a Windows Server 2003, x64 Edition platform, users must have “Read” and “Execute” privileges for `cmd.exe`.

- ◆ Setting the `JOB_STARTER` parameter in `lsb.queues` to `$USER_STARTER` enables users to define their own job starters by defining the environment variable `USER_STARTER`.
- ◆ Setting a job starter to `make clean` causes the command `make clean` to be run before the user job.

Command-Level Job Starters

A command-level job starter allows you to specify an executable file that does any necessary setup for the job and runs the job when the setup is complete. You can select an existing command to be a job starter, or you can create a script containing a desired set of commands to serve as a job starter.

This section describes how to set up and use a command-level job starter to run interactive jobs.

Command-level job starters have no effect on batch jobs, including interactive batch jobs. See Chapter 39, “[Interactive Jobs with bsub](#)” for information on interactive batch jobs.

A job starter can also be defined at the queue level using the `JOB_STARTER` parameter. Only the LSF administrator can configure queue-level job starters. See [Queue-Level Job Starters](#) on page 540 for more information.

LSF_JOB_STARTER environment variable

Use the LSF_JOB_STARTER environment variable to specify a command or script that is the job starter for the interactive job. When the environment variable LSF_JOB_STARTER is defined, RES invokes the job starter rather than running the job itself, and passes the job to the job starter as a command-line argument.

Using command-level job starters

UNIX

The job starter is invoked from within a Bourne shell, making the command-line equivalent:

```
/bin/sh -c "$LSF_JOB_STARTER command [argument ...]"
```

where *command* and *argument* are the command-line arguments you specify in `lshrun`, `lshgrun`, or `ch`.

Windows

RES runs the job starter, passing it your commands as arguments:

```
LSF_JOB_STARTER command [argument ...]
```

Examples

UNIX

If you define the LSF_JOB_STARTER environment variable using the following C-shell command:

```
% setenv LSF_JOB_STARTER "/bin/sh -c"
```

Then you run a simple C-shell job:

```
% lshrun "'a.out; hostname'"
```

The command that actually runs is:

```
/bin/sh -c "/bin/sh -c 'a.out hostname'"
```

The job starter can be a shell script. In the following example, the LSF_JOB_STARTER environment variable is set to the Bourne shell script named `job_starter`:

```
$ LSF_JOB_STARTER=/usr/local/job_starter
```

The `job_starter` script contains the following:

```
#!/bin/sh
set term = xterm
eval "$*"

```

Windows

If you define the LSF_JOB_STARTER environment variable as follows:

```
set LSF_JOB_STARTER=C:\cmd.exe /C
```

Then you run a simple DOS shell job:

```
C:\> lshrun dir /p
```

The command that actually runs is:

```
C:\cmd.exe /C dir /p
```

Queue-Level Job Starters

LSF administrators can define a job starter for an individual queue to create a specific environment for jobs to run in. A queue-level job starter specifies an executable that performs any necessary setup, and then runs the job when the setup is complete. The `JOB_STARTER` parameter in `lsb.queues` specifies the command or script that is the job starter for the queue.

This section describes how to set up and use a queue-level job starter.

Queue-level job starters have no effect on interactive jobs, unless the interactive job is submitted to a queue as an interactive batch job. See Chapter 39, “[Interactive Jobs with `bsub`](#)” for information on interactive batch jobs.

LSF users can also select an existing command or script to be a job starter for their interactive jobs using the `LSF_JOB_STARTER` environment variable. See [Command-Level Job Starters](#) on page 538 for more information.

Configure a queue-level job starter

- 1 Use the `JOB_STARTER` parameter in `lsb.queues` to specify a queue-level job starter in the queue definition. All jobs submitted to this queue are run using the job starter. The jobs are called by the specified job starter process rather than initiated by the batch daemon process.

For example:

```
Begin Queue
.
JOB_STARTER = xterm -e
.
End Queue
```

All jobs submitted to this queue are run under an `xterm` terminal emulator.

`JOB_STARTER` parameter (`lsb.queues`)

The `JOB_STARTER` parameter in the queue definition (`lsb.queues`) has the following format:

```
JOB_STARTER=starter [starter] ["%USR CMD"] [starter]
```

The string *starter* is the command or script that is used to start the job. It can be any executable that can accept a job as an input argument. Optionally, additional strings can be specified.

When starting a job, LSF runs the `JOB_STARTER` command, and passes the shell script containing the job commands as the argument to the job starter. The job starter is expected to do some processing and then run the shell script containing the job commands. The command is run under `/bin/sh -c` and can contain any valid Bourne shell syntax.

%USRCMD string The special string `%USRCMD` indicates the position of the job starter command in the job command line. By default, the user commands run after the job starter, so the `%USRCMD` string is not usually required. For example, these two job starters both give the same results:

```
JOB_STARTER = /bin/csh -c
JOB_STARTER = /bin/csh -c "%USRCMD"
```

You must enclose the `%USRCMD` string in quotes. The `%USRCMD` string can be followed by additional commands. For example:

```
JOB_STARTER = /bin/csh -c "%USRCMD;sleep 10"
```

If a user submits the following job to the queue with this job starter:

```
bsub myjob arguments
```

the command that actually runs is:

```
/bin/csh -c "myjob arguments; sleep 10"
```

For more information

See the *Platform LSF Configuration Reference* for information about the `JOB_STARTER` parameter in the `lsb.queues` file.

Controlling Execution Environment Using Job Starters

In some cases, using `bsub -L` does not result in correct environment settings on the execution host. LSF provides the following two job starters:

- ◆ `preservestarter` — preserves the default environment of the execution host. It does not include any submission host settings.
- ◆ `augmentstarter` — augments the default user environment of the execution host by adding settings from the submission host that are not already defined on the execution host

`bsub -L` cannot be used for a Windows execution host.

Where the job starter executables are located

By default, the job starter executables are installed in `LSF_BINDIR`. If you prefer to store them elsewhere, make sure they are in a directory that is included in the default `PATH` on the execution host.

For example:

- ◆ On Windows, put the job starter under `%WINDIR%`.
- ◆ On UNIX, put the job starter under `$HOME/bin`.

Source code for the job starters

The source code for the job starters is installed in `LSF_MISC/examples`.

Adding to the initial login environment

By default, the `preservestarter` job starter preserves the environment that RES establishes on the execution host, and establishes an initial login environment for the user with the following variables from the user's login environment on the execution host:

- ◆ HOME
- ◆ USER

Controlling Execution Environment Using Job Starters

- ◆ SHELL
- ◆ LOGNAME

Any additional environment variables that exist in the user's login environment on the submission host must be added to the job starter source code.

Example

A user's `.login` script on the submission host contains the following setting:

```
if ($TERM != "xterm") then
    set TERM=`tset - -Q -m 'switch:?vt100' ....
else
    stty -tabs
endif
```

The `TERM` environment variable must also be included in the environment on the execution host for login to succeed. If it is missing in the job starter, the login fails, the job starter may fail as well. If the job starter can continue with only the initial environment settings, the job may execute correctly, but this is not likely.

External Job Submission and Execution Controls

This document describes the use of external job submission and execution controls called `esub` and `eexec`. These site-specific user-written executables are used to validate, modify, and reject job submissions, pass data to and modify job execution environments.

Contents

- ◆ [Understanding External Executables](#) on page 543
- ◆ [Using `esub`](#) on page 544
- ◆ [Working with `eexec`](#) on page 552

Understanding External Executables

About `esub` and `eexec`

LSF provides the ability to validate, modify, or reject job submissions, modify execution environments, and pass data from the submission host directly to the execution host through the use of the `esub` and `eexec` executables. Both are site-specific and user written and must be located in `LSF_SERVERDIR`.

Validate, modify, or reject a job

To validate, modify, or reject a job, an `esub` needs to be written. See [Using `esub`](#) on page 544

Modifying execution environments

To modify the execution environment on the execution host, an `eexec` needs to be written. See [Working with `eexec`](#) on page 552

Passing data

To pass data directly to the execution host, an `esub` and `eexec` need to be written. See [Using `esub` and `eexec` to pass data to execution environments](#) on page 552

Interactive remote execution

Interactive remote execution also runs `esub` and `eexec` if they are found in `LSF_SERVERDIR`. For example, `lsrun` invokes `esub`, and `RES` runs `eexec` before starting the task. `esub` is invoked at the time of the `ls_connect(3)` call, and `RES` invokes `eexec` each time a remote task is executed. `RES` runs `eexec` only at task startup time.

DCE credentials and AFS tokens

`esub` and `eexec` are also used for processing DCE credentials and AFS tokens. See the following documents on the Platform Web site for more information:

- ◆ “Installing LSF on AFS”
- ◆ “Installing LSF on DCE/DFS”

Using esub

About esub

An `esub`, short for *external submission*, is a user-written executable (binary or script) that can be used to validate, modify, or reject jobs. The `esub` is put into `LSF_SERVERDIR` (defined in `lsf.conf`) where LSF checks for its existence when a job is submitted, restarted, and modified. If LSF finds an `esub`, it is run by LSF. Whether the job is submitted, modified, or rejected depends on the logic built into the `esub`.

Any messages that need to be provided to the user should be directed to the standard error (`stderr`) stream and not the standard output (`stdout`) stream.

In this section

- ◆ [Environment variables to bridge esub and LSF](#) on page 544
- ◆ [General esub logic](#) on page 547
- ◆ [Rejecting jobs](#) on page 548
- ◆ [Validating job submission parameters](#) on page 548
- ◆ [Modifying job submission parameters](#) on page 549
- ◆ [Using bmod and brestart commands with mesub](#) on page 549
- ◆ [Use multiple esub \(mesub\)](#) on page 550
- ◆ [How master esub invokes application-specific esubs](#) on page 550
- ◆ [Configure master esub and your application-specific esub](#) on page 551

Environment variables to bridge esub and LSF

LSF provides the following environment variables in the `esub` execution environment:

LSB_SUB_PARM_FILE

This variable points to a temporary file containing the job parameters that `esub` reads when the job is submitted. The submission parameters are a set of name-value pairs on separate lines in the format "`option_name=value`".

The following option names are supported:

Option	Description
LSB_SUB_ADDITIONAL	String format parameter containing the value of the <code>-a</code> option to <code>bsub</code> The value of <code>-a</code> is passed to <code>esub</code> , but it does not directly affect the other <code>bsub</code> parameters or behavior. The value of <code>-a</code> must correspond to an actual <code>esub</code> file. For example, to use <code>bsub -a fluent</code> , the file <code>esub.fluent</code> must exist in <code>LSF_SERVERDIR</code> . LSB_SUB_ADDITIONAL cannot be changed in or added to <code>LSB_SUB_MODIFY_FILE</code> .
LSB_SUB_BEGIN_TIME	Begin time, in seconds since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_CHKPNT_DIR	Checkpoint directory
LSB_SUB_COMMAND_LINE	<code>bsub</code> job command argument LSB_SUB_COMMANDNAME must be set in <code>lsf.conf</code> to enable <code>esub</code> to use this variable
LSB_SUB_CHKPNT_PERIOD	Checkpoint period in seconds
LSB_SUB_DEPEND_COND	Dependency condition
LSB_SUB_ERR_FILE	Standard error file name
LSB_SUB_EXCEPTION	Exception condition
LSB_SUB_EXCLUSIVE	"Y" specifies exclusive execution
LSB_SUB_EXTSCHED_PARAM	Validate or modify <code>bsub -extsched</code> option
LSB_SUB_HOLD	Hold job (<code>bsub -H</code> option)
LSB_SUB_HOSTS	List of execution host names
LSB_SUB_HOST_SPEC	Host specifier
LSB_SUB_IN_FILE	Standard input file name
LSB_SUB_INTERACTIVE	"Y" specifies an interactive job
LSB_SUB_LOGIN_SHELL	Login shell
LSB_SUB_JOB_NAME	Job name
LSB_SUB_JOB_WARNING_ACTION	Job warning action specified by <code>bsub -wa</code>
LSB_SUB_JOB_ACTION_WARNING_TIME	Job warning time period specified by <code>bsub -wt</code>
LSB_SUB_MAIL_USER	Email address used by LSF for sending job email
LSB_SUB_MAX_NUM_PROCESSORS	Maximum number of processors requested
LSB_SUB_MODIFY	"Y" specifies a modification request
LSB_SUB_MODIFY_ONCE	"Y" specifies a modification-once request
LSB_SUB_NOTIFY_BEGIN	"Y" specifies email notification when job begins
LSB_SUB_NOTIFY_END	"Y" specifies email notification when job ends
LSB_SUB_NUM_PROCESSORS	Minimum number of processors requested
LSB_SUB_OTHER_FILES	The value is <code>SUB_RESET</code> if defined to indicate a <code>bmod</code> is being performed to reset the number of files to be transferred
LSB_SUB_OTHER_FILES_ <i>number</i>	<i>number</i> is an index number indicating the particular file transfer value is the specified file transfer expression. For example, for <code>bsub -f "a > b" -f "c < d"</code> , the following would be defined: LSB_SUB_OTHER_FILES_0="a > b" LSB_SUB_OTHER_FILES_1="c < d"
LSB_SUB_OUT_FILE	Standard output file name
LSB_SUB_PRE_EXEC	Pre-execution command

Using esub

Option	Description
LSB_SUB_PROJECT_NAME	Project name
LSB_SUB_PTY	"Y" specifies an interactive job with PTY support
LSB_SUB_PTY_SHELL	"Y" specifies an interactive job with PTY shell support
LSB_SUB_QUEUE	Submission queue name
LSB_SUB_RERUNNABLE	"Y" specifies a rerunnable job "N" specifies a nonrerunnable job (specified with bsub -rn). The job is not rerunnable even it was submitted to a rerunable queue or application profile For bmod -rn, the value is SUB_RESET.
LSB_SUB_RES_REQ	Resource requirement string— <i>does not</i> support multiple resource requirement strings
LSB_SUB_RESTART	"Y" specifies a restart job
LSB_SUB_RESTART_FORCE	"Y" specifies forced restart job
LSB_SUB_RLIMIT_CORE	Core file size limit
LSB_SUB_RLIMIT_CPU	CPU limit
LSB_SUB_RLIMIT_DATA	Data size limit
LSB_SUB_RLIMIT_FSIZE	File size limit
LSB_SUB_RLIMIT_PROCESS	Process limit
LSB_SUB_RLIMIT_RSS	Resident size limit
LSB_SUB_RLIMIT_RUN	Wall-clock run limit
LSB_SUB_RLIMIT_STACK	Stack size limit
LSB_SUB_RLIMIT_SWAP	Virtual memory limit (swap space)
LSB_SUB_RLIMIT_THREAD	Thread limit
LSB_SUB_TERM_TIME	Termination time, in seconds, since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_TIME_EVENT	Time event expression
LSB_SUB_USER_GROUP	User group name
LSB_SUB_WINDOW_SIG	Window signal number
LSB_SUB2_JOB_GROUP	Options specified by bsub -g
LSB_SUB2_LICENSE_PROJECT	LSF License Scheduler project name specified by bsub -Lp
LSB_SUB2_IN_FILE_SPOOL	Spooled input file (bsub -is)
LSB_SUB2_JOB_CMD_SPOOL	Spooled job command file (bsub -Zs)
LSB_SUB2_JOB_PRIORITY	Job priority (bsub- sp and bmod -sp) For bmod -spn, the value is SUB_RESET
LSB_SUB2_SLA	SLA scheduling options
LSB_SUB2_USE_RSV	Advance reservation ID specified by bsub -U
LSB_SUB3_ABSOLUTE_PRIORITY	For bmod -aps, the value equal to the APS string given with the bmod -aps. For bmod -apsn, the value is SUB_RESET.
LSB_SUB3_APP	Options specified by bsub- app and bmod -app. For bmod -appn, the value is SUB_RESET.
LSB_SUB3_CWD	Current working directory specified on on the command line with bsub -cwd
LSB_SUB3_POST_EXEC	Run the specified post-execution command on the execution host after the job finishes. Specified by bsub -Ep.
LSB_SUB3_RUNTIME_ESTIMATION	Runtime estimate spedified by bsub -We
LSB_SUB3_USER_SHELL_LIMITS	Pass user shell limits to execution host. Spedified by bsub -ul.

Example submission parameter file

If a user submits the following job:

```
bsub -q normal -x -P my_project -R "r1m rusage[dummy=1]" -n 90 sleep
10
```

The contents of the LSB_SUB_PARM_FILE will be:

```
LSB_SUB_QUEUE="normal"
LSB_SUB_EXCLUSIVE=Y
LSB_SUB_RES_REQ="r1m rusage[dummy=1]"
LSB_SUB_PROJECT_NAME="my_project"
LSB_SUB_COMMAND_LINE="sleep 10"
LSB_SUB_NUM_PROCESSORS=90
LSB_SUB_MAX_NUM_PROCESSORS=90
```

LSB_SUB_ABORT_VALUE

This variable indicates the value `esub` should exit with if LSF is to reject the job submission.

LSB_SUB_MODIFY_ENVFILE

The file in which `esub` should write any changes to the job environment variables. `esub` writes the variables to be modified to this file in the same format used in `LSB_SUB_PARM_FILE`. The order of the variables does not matter.

After `esub` runs, LSF checks `LSB_SUB_MODIFY_ENVFILE` for changes and if found, LSF will apply them to the job environment variables.

LSB_SUB_MODIFY_FILE

The file in which `esub` should write any submission parameter changes.

`esub` writes the job options to be modified to this file in the same format used in `LSB_SUB_PARM_FILE`. The order of the options does not matter. After `esub` runs, LSF checks `LSB_SUB_MODIFY_FILE` for changes and if found LSF will apply them to the job.

TIP: `LSB_SUB_ADDITIONAL` cannot be changed in or added to `LSB_SUB_MODIFY_FILE`.

LSF_INVOKE_CMD

Indicates the name of the last LSF command that invoked an external executable (for example, `esub`).

External executables get called by several LSF commands (`bsub`, `bmod`, `lsrcun`). This variable contains the name of the last LSF command to call the executable.

General esub logic

After `esub` runs, LSF checks:

- 1 Is the `esub` exit value `LSB_SUB_ABORT_VALUE`?
 - a Yes, step 2
 - b No, step 4
- 2 Reject the job

Using esub

- 3 Go to step 5
- 4 Does LSB_SUB_MODIFY_FILE or LSB_SUB_MODIFY_ENVFILE exist?
 - ❖ Apply changes
- 5 Done

Rejecting jobs

Depending on your policies you may choose to reject a job. To do so, have `esub` exit with `LSB_SUB_ABORT_VALUE`.

If `esub` rejects the job, it should not write to either `LSB_SUB_MODIFY_FILE` or `LSB_SUB_MODIFY_ENVFILE`.

Example

The following Bourne shell `esub` rejects all job submissions by exiting with `LSB_SUB_ABORT_VALUE`:

```
#!/bin/sh

# Redirect stderr to stdout so echo can be used for
# error messages
exec 1>&2

# Reject the submission
echo "LSF is Rejecting your job submission..."
exit $LSB_SUB_ABORT_VALUE
```

Validating job submission parameters

One use of validation is to support project-based accounting. The user can request that the resources used by a job be charged to a particular project. Projects are associated with a job at job submission time, so LSF will accept any arbitrary string for a project name. In order to ensure that only valid projects are entered and the user is eligible to charge to that project, an `esub` can be written.

Example

The following Bourne shell `esub` validates job submission parameters:

```
#!/bin/sh

. $LSB_SUB_PARM_FILE

# Redirect stdout to stderr so echo can be used for error messages
exec 1>&2

# Check valid projects
if [ $LSB_SUB_PROJECT_NAME != "proj1" -o $LSB_SUB_PROJECT_NAME != "proj2" ]; then
    echo "Incorrect project name specified"
    exit $LSB_SUB_ABORT_VALUE
fi

USER=`whoami`
if [ $LSB_SUB_PROJECT_NAME = "proj1" ]; then
    # Only user1 and user2 can charge to proj1
    if [ $USER != "user1" -a $USER != "user2" ]; then
        echo "You are not allowed to charge to this project"
```

```

        exit $LSB_SUB_ABORT_VALUE
    fi
fi

```

Modifying job submission parameters

`esub` can be used to modify submission parameters and the job environment before the job is actually submitted.

The following example writes modifications to `LSB_SUB_MODIFY_FILE` for the following parameters:

- ◆ `LSB_SUB_QUEUE`
- ◆ `USER`
- ◆ `SHELL`

In the example, user `userA` can only submit jobs to queue `queueA`. User `userB` must use Bourne shell (`/bin/sh`), and user `userC` should never be able to submit a job.

```

#!/bin/sh
. $LSB_SUB_PARM_FILE

# Redirect stderr to stdout so echo can be used for error messages
exec 1>&2

USER=`whoami`
# Ensure userA is using the right queue queueA
if [ $USER="userA" -a $LSB_SUB_QUEUE != "queueA" ]; then
    echo "userA has submitted a job to an incorrect queue"
    echo "...submitting to queueA"
    echo 'LSB_SUB_QUEUE="queueA"' > $LSB_SUB_MODIFY_FILE
fi

# Ensure userB is using the right shell (/bin/sh)
if [ $USER="userB" -a $SHELL != "/bin/sh" ]; then
    echo "userB has submitted a job using $SHELL"
    echo "...using /bin/sh instead"
    echo 'SHELL="/bin/sh"' > $LSB_SUB_MODIFY_ENVFILE
fi

# Deny userC the ability to submit a job
if [ $USER="userC" ]; then
    echo "You are not permitted to submit a job."
    exit $LSB_SUB_ABORT_VALUE
fi

```

Using `bmod` and `brestart` commands with `mesub`

You can use the `bmod` command to modify job submission parameters, and `brestart` to restart checkpointed jobs. Like `bsub`, `bmod` and `brestart` also call `mesub`, which in turn invoke any existing `esub` executables in `LSF_SERVERDIR`. `bmod` and `brestart` cannot make changes to the job environment through `mesub` and `esub`. Environment changes only occur when `mesub` is called by the original job submission with `bsub`.

Use multiple esub (mesub)

LSF provides a master esub (LSF_SERVERDIR/mesub) to handle the invocation of individual application-specific esub executables and the job submission requirements of your applications.

- 1 Use the `-a` option of `bsub` to specify the application you are running through LSF.

For example, to submit a FLUENT job:

```
bsub -a fluent bsub_options fluent_command
```

The method name `fluent`, uses the esub for FLUENT jobs (LSF_SERVERDIR/esub.fluent), which sets the checkpointing method `LSB_ECHKPNT_METHOD="fluent"` to use the `echkpnt.fluent` and `erestart.fluent`.

LSB_ESUB_METHOD (lsf.conf)

To specify a mandatory esub method that applies to all job submissions, you can configure `LSB_ESUB_METHOD` in `lsf.conf`.

`LSB_ESUB_METHOD` specifies the name of the esub method used in addition to any methods specified in the `bsub -a` option.

For example, `LSB_ESUB_METHOD="dce fluent"` defines DCE as the mandatory security system, and FLUENT as the mandatory application used on all jobs.

Compatibility note

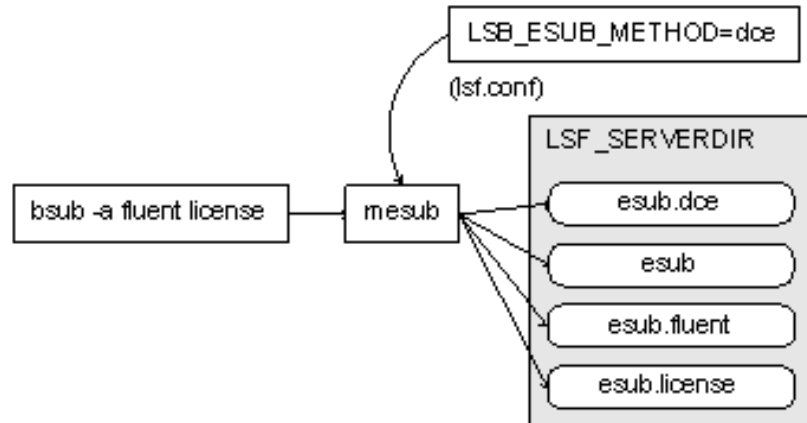
RESTRICTION: After LSF version 5.1, the value of `-a` and `LSB_ESUB_METHOD` must correspond to an actual esub file in LSF_SERVERDIR. For example, to use `bsub -a fluent`, the file `esub.fluent` must exist in LSF_SERVERDIR.

How master esub invokes application-specific esubs

`bsub` invokes `mesub` at job submission, which calls esub programs in this order:

- 1 Mandatory esub programs defined by `LSB_ESUB_METHOD`
- 2 Any existing executable named `LSF_SERVERDIR/esub`
- 3 Application-specific esub programs in the order specified in the `bsub -a` option

Example



In this example:

- ◆ `esub.dce` is defined as the only mandatory `esub`
- ◆ An executable named `esub` already exists in `LSF_SERVERDIR`
- ◆ Executables named `esub.fluent` and `esub.license` exist in `LSF_SERVERDIR`
- ◆ `bsub -a fluent license` submits the job as a FLUENT job, and `mesub` invokes the following `esub` executables in `LSF_SERVERDIR` in this order:
 - ❖ `esub.dce`
 - ❖ `esub`
 - ❖ `esub.fluent`
 - ❖ `esub.license`
- ◆ `bsub` without the `-a` option submits the job, and `mesub` invokes only the mandatory `esub.dce` and the existing `esub` in `LSF_SERVERDIR`, not the application-specific `esub` programs.

Configure master `esub` and your application-specific `esub`

The master `esub` is installed as `LSF_SERVERDIR/mesub`. After installation:

- 1 Create your own application-specific `esub`.
- 2 Optional. Configure `LSB_ESUB_METHOD` in `lsf.conf` to specify a mandatory `esub` for all job submissions.

Name your `esub`

- 1 Use the following naming conventions:
 - ❖ On UNIX, `LSF_SERVERDIR/esub.application`
 - ❖ On Windows, `LSF_SERVERDIR\esub.application.[exe |bat]`

For FLUENT jobs, for example:

- ◆ UNIX: `esub.fluent`
- ◆ Windows: `esub.fluent.exe`

Existing esub

The name of the `esub` program must be a valid file name. It can contain only alphanumeric characters, underscore (`_`) and hyphen (`-`).

CAUTION: The file name `esub.user` is reserved for backward compatibility. Do not use the name `esub.user` for your application-specific `esub`.

Existing esub

Your existing `esub` does not need to follow this convention and does not need to be renamed. However, since `mesub` invokes any `esub` that follows this convention, you should move any backup copies of your `esubs` out of `LSF_SERVERDIR` or choose a name that does not follow the convention (for example, use `esub_bak` instead of `esub.bak`).

Working with eexec

About eexec

The `eexec` program runs on the execution host at job start-up and completion time and when checkpointing is initiated. It is run as the user after the job environment variables have been set. The environment variable `LSF_EXEC_T` is set to `START`, `END`, and `CHKPNT`, respectively, to indicate when `eexec` is invoked.

If you need to run `eexec` as a different user, such as `root`, you must properly define `LSF_EEXEC_USER` in the file `/etc/lsf.sudoers`. See the *Platform LSF Configuration Reference* for information about the `lsf.sudoers` file.

`eexec` is expected to finish running because the parent job process waits for `eexec` to finish running before proceeding. The environment variable `LSF_JOBPID` stores the process ID of the process that invoked `eexec`. If `eexec` is intended to monitor the execution of the job, `eexec` must fork a child and then have the parent `eexec` process exit. The `eexec` child should periodically test that the job process is still alive using the `LSF_JOBPID` variable.

Using esub and eexec to pass data to execution environments

If `esub` needs to pass some data to `eexec`, it can write the data to its standard output for `eexec` to read from its standard input (`stdin`). LSF effectively acts as the pipe between `esub` and `eexec` (e.g., `esub | eexec`).

Standard output (`stdout`) from any `esub` is automatically sent to `eexec`.

Limitation

Since `eexec` cannot handle more than one standard output stream, only one `esub` can use standard output to generate data as standard input to `eexec`.

For example, the `esub` for AFS (`esub.afs`) sends its authentication tokens as standard output to `eexec`. If you use AFS, no other `esub` can use standard output.

Configuring Job Controls

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change. This chapter describes how to configure job control actions to override or augment the default job control actions.

Contents

- ◆ [Default Job Control Actions](#) on page 553
- ◆ [Configuring Job Control Actions](#) on page 555
- ◆ [Customizing Cross-Platform Signal Conversion](#) on page 558

Default Job Control Actions

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change. LSF supports the following default actions for job controls:

- ◆ SUSPEND
- ◆ RESUME
- ◆ TERMINATE

On successful completion of the job control action, the LSF job control commands cause the status of a job to change.

The environment variable `LS_EXEC_T` is set to the value `JOB_CONTROLS` for a job when a job control action is initiated.

See [Killing Jobs](#) on page 112 for more information about job controls and the LSF commands that perform them.

SUSPEND action

Change a running job from RUN state to one of the following states:

- ◆ USUSP or PSUSP in response to `bstop`
- ◆ SSUSP state when the LSF system suspends the job

The default action is to send the following signals to the job:

Default Job Control Actions

- ◆ SIGTSTP for parallel or interactive jobs. SIGTSTP is caught by the master process and passed to all the slave processes running on other hosts.
- ◆ SIGSTOP for sequential jobs. SIGSTOP cannot be caught by user programs. The SIGSTOP signal can be configured with the LSB_SIGSTOP parameter in `lsf.conf`.

LSF invokes the SUSPEND action when:

- ◆ The user or LSF administrator issues a `bstop` or `bkill` command to the job
- ◆ Load conditions on the execution host satisfy *any* of:
 - ❖ The suspend conditions of the queue, as specified by the STOP_COND parameter in `lsb.queues`
 - ❖ The scheduling thresholds of the queue or the execution host
- ◆ The run window of the queue closes
- ◆ The job is preempted by a higher priority job

RESUME action

Change a suspended job from SSUSP, USUSP, or PSUSP state to the RUN state. The default action is to send the signal SIGCONT.

LSF invokes the RESUME action when:

- ◆ The user or LSF administrator issues a `bresume` command to the job
- ◆ Load conditions on the execution host satisfy *all* of:
 - ❖ The resume conditions of the queue, as specified by the RESUME_COND parameter in `lsb.queues`
 - ❖ The scheduling thresholds of the queue and the execution host
- ◆ A closed run window of the queue opens again
- ◆ A preempted job finishes

TERMINATE action

Terminate a job. This usually causes the job change to EXIT status. The default action is to send SIGINT first, then send SIGTERM 10 seconds after SIGINT, then send SIGKILL 10 seconds after SIGTERM. The delay between signals allows user programs to catch the signals and clean up before the job terminates.

To override the 10 second interval, use the parameter `JOB_TERMINATE_INTERVAL` in the `lsb.params` file. See the *Platform LSF Configuration Reference* for information about the `lsb.params` file.

LSF invokes the TERMINATE action when:

- ◆ The user or LSF administrator issues a `bkill` or `bqueue` command to the job
- ◆ The TERMINATE_WHEN parameter in the queue definition (`lsb.queues`) causes a SUSPEND action to be redirected to TERMINATE
- ◆ The job reaches its CPULIMIT, MEMLIMIT, RUNLIMIT or PROCESSLIMIT

If the execution of an action is in progress, no further actions are initiated unless it is the TERMINATE action. A TERMINATE action is issued for all job states except PEND.

Windows job control actions

On Windows, actions equivalent to the UNIX signals have been implemented to do the default job control actions. Job control messages replace the SIGINT and SIGTERM signals, but only customized applications will be able to process them. Termination is implemented by the `TerminateProcess()` system call.

See *Platform LSF Programmer's Guide* for more information about LSF signal handling on Windows.

Configuring Job Control Actions

Several situations may require overriding or augmenting the default actions for job control. For example:

- ◆ Notifying users when their jobs are suspended, resumed, or terminated
- ◆ An application holds resources (for example, licenses) that are not freed by suspending the job. The administrator can set up an action to be performed that causes the license to be released before the job is suspended and re-acquired when the job is resumed.
- ◆ The administrator wants the job checkpointed before being:
 - ❖ Suspended when a run window closes
 - ❖ Killed when the RUNLIMIT is reached
- ◆ A distributed parallel application must receive a catchable signal when the job is suspended, resumed or terminated to propagate the signal to remote processes.

To override the default actions for the SUSPEND, RESUME, and TERMINATE job controls, specify the `JOB_CONTROLS` parameter in the queue definition in `lsb.queues`.

See the *Platform LSF Configuration Reference* for information about the `lsb.queues` file.

JOB_CONTROLS parameter (lsb.queues)

The `JOB_CONTROLS` parameter has the following format:

```
Begin Queue
...
JOB_CONTROLS = SUSPEND[signal | CHPNT | command] \
                RESUME[signal | command] \
                TERMINATE[signal | CHPNT | command]
...
End Queue
```

When LSF needs to suspend, resume, or terminate a job, it invokes one of the following actions as specified by SUSPEND, RESUME, and TERMINATE.

signal

A UNIX signal name (for example, SIGTSTP or SIGTERM). The specified signal is sent to the job.

The same set of signals is not supported on all UNIX systems. To display a list of the symbolic names of the signals (without the SIG prefix) supported on your system, use the `kill -l` command.

CHKPNT

Checkpoint the job. Only valid for SUSPEND and TERMINATE actions.

- ◆ If the SUSPEND action is CHKPNT, the job is checkpointed and then stopped by sending the SIGSTOP signal to the job automatically.
- ◆ If the TERMINATE action is CHKPNT, then the job is checkpointed and killed automatically.

command

A `/bin/sh` command line.

- ◆ Do not quote the command line inside an action definition.
- ◆ Do not specify a signal followed by an action that triggers the same signal (for example, do not specify `JOB_CONTROLS=TERMINATE[bkill]` or `JOB_CONTROLS=TERMINATE[brequeue]`). This will cause a deadlock between the signal and the action.

Using a command as a job control action

- ◆ The command line for the action is run with `/bin/sh -c` so you can use shell features in the command.
- ◆ The command is run as the user of the job.
- ◆ All environment variables set for the job are also set for the command action. The following additional environment variables are set:
 - ❖ `LSB_JOBPGIDS`—a list of current process group IDs of the job
 - ❖ `LSB_JOBPIIDS`—a list of current process IDs of the job
- ◆ For the SUSPEND action command, the environment variables `LSB_SUSP_REASONS` and `LSB_SUSP_SUBREASONS` are also set. Use them together in your custom job control to determine the exact load threshold that caused a job to be suspended.
 - ❖ `LSB_SUSP_REASONS`—an integer representing a bitmap of suspending reasons as defined in `lsbatch.h`. The suspending reason can allow the command to take different actions based on the reason for suspending the job.
 - ❖ `LSB_SUSP_SUBREASONS`—an integer representing the load index that caused the job to be suspended. When the suspending reason `SUSP_LOAD_REASON` (suspended by load) is set in `LSB_SUSP_REASONS`, `LSB_SUSP_SUBREASONS` is set to one of the load index values defined in `lsf.h`.
- ◆ The standard input, output, and error of the command are redirected to the NULL device, so you cannot tell directly whether the command runs correctly. The default null device on UNIX is `/dev/null`.
- ◆ You should make sure the command line is correct. If you want to see the output from the command line for testing purposes, redirect the output to a file inside the command line.

TERMINATE job actions

Use caution when configuring TERMINATE job actions that do more than just kill a job. For example, resource usage limits that terminate jobs change the job state to SSUSP while LSF waits for the job to end. If the job is not killed by the TERMINATE action, it remains suspended indefinitely.

TERMINATE_WHEN parameter (lsb.queues)

In certain situations you may want to terminate the job instead of calling the default SUSPEND action. For example, you may want to kill jobs if the run window of the queue is closed. Use the TERMINATE_WHEN parameter to configure the queue to invoke the TERMINATE action instead of SUSPEND.

See the *Platform LSF Configuration Reference* for information about the `lsb.queues` file and the TERMINATE_WHEN parameter.

Syntax

```
TERMINATE_WHEN = [LOAD] [PREEMPT] [WINDOW]
```

Example

The following defines a night queue that will kill jobs if the run window closes.

```
Begin Queue
NAME          = night
RUN_WINDOW    = 20:00-08:00
TERMINATE_WHEN = WINDOW
JOB_CONTROLS  = TERMINATE[ kill -KILL $LSB_JOBPIIDS;
                  echo "job $LSB_JOBID killed by queue run window" |
                  mail $USER ]
End Queue
```

LSB_SIGSTOP parameter (lsf.conf)

Use LSB_SIGSTOP to configure the SIGSTOP signal sent by the default SUSPEND action.

If LSB_SIGSTOP is set to anything other than SIGSTOP, the SIGTSTP signal that is normally sent by the SUSPEND action is not sent. For example, if `LSB_SIGSTOP=SIGKILL`, the three default signals sent by the TERMINATE action (SIGINT, SIGTERM, and SIGKILL) are sent 10 seconds apart.

See the *Platform LSF Configuration Reference* for information about the `lsf.conf` file.

Avoiding signal and action deadlock

Do not configure a job control to contain the signal or command that is the same as the action associated with that job control. This will cause a deadlock between the signal and the action.

For example, the `kill` command uses the TERMINATE action, so a deadlock results when the TERMINATE action itself contains the `kill` command.

Any of the following job control specifications will cause a deadlock:

- ◆ `JOB_CONTROLS=TERMINATE[kill]`
- ◆ `JOB_CONTROLS=TERMINATE[brequeue]`
- ◆ `JOB_CONTROLS=RESUME[bresume]`
- ◆ `JOB_CONTROLS=SUSPEND[bstop]`

Customizing Cross-Platform Signal Conversion

LSF supports signal conversion between UNIX and Windows for remote interactive execution through RES.

On Windows, the CTRL+C and CTRL+BREAK key combinations are treated as signals for console applications (these signals are also called console control actions).

LSF supports these two Windows console signals for remote interactive execution. LSF regenerates these signals for user tasks on the execution host.

Default signal conversion

In a mixed Windows/UNIX environment, LSF has the following default conversion between the Windows console signals and the UNIX signals:

Windows	UNIX
CTRL+C	SIGINT
CTRL+BREAK	SIGQUIT

For example, if you issue the `lshrun` or `bsub -I` commands from a Windows console but the task is running on an UNIX host, pressing the CTRL+C keys will generate a UNIX SIGINT signal to your task on the UNIX host. The opposite is also true.

Custom signal conversion

For `lshrun` (but not `bsub -I`), LSF allows you to define your own signal conversion using the following environment variables:

- ◆ LSF_NT2UNIX_CLTRC
- ◆ LSF_NT2UNIX_CLTRB

For example:

- ◆ LSF_NT2UNIX_CLTRC=SIGXXXX
- ◆ LSF_NT2UNIX_CLTRB=SIGYYYY

Here, SIGXXXX/SIGYYYY are UNIX signal names such as SIGQUIT, SIGTINT, etc. The conversions will then be: CTRL+C=SIGXXXX and CTRL+BREAK=SIGYYYY.

If both LSF_NT2UNIX_CLTRC and LSF_NT2UNIX_CLTRB are set to the same value (LSF_NT2UNIX_CLTRC=SIGXXXX and LSF_NT2UNIX_CLTRB=SIGXXXX), CTRL+C will be generated on the Windows execution host.

For `bsub -I`, there is no conversion other than the default conversion.

VI

Interactive Jobs

- ◆ [Interactive Jobs with bsub](#) on page 561
- ◆ [Running Interactive and Remote Tasks](#) on page 573

Interactive Jobs with bsub

Contents

- ◆ [About Interactive Jobs](#) on page 561
- ◆ [Submitting Interactive Jobs](#) on page 562
- ◆ [Performance Tuning for Interactive Batch Jobs](#) on page 564
- ◆ [Interactive Batch Job Messaging](#) on page 567
- ◆ [Running X Applications with bsub](#) on page 568
- ◆ [Writing Job Scripts](#) on page 568
- ◆ [Registering utmp File Entries for Interactive Batch Jobs](#) on page 571

About Interactive Jobs

It is sometimes desirable from a system management point of view to control all workload through a single centralized scheduler.

Running an interactive job through the LSF batch system allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs. You can submit a job and the least loaded host is selected to run the job.

Since all interactive batch jobs are subject to LSF policies, you will have more control over your system. For example, you may dedicate two servers as interactive servers, and disable interactive access to all other servers by defining an interactive queue that only uses the two interactive servers.

Scheduling policies

Running an interactive batch job allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs.

An interactive batch job is scheduled using the same policy as all other jobs in a queue. This means an interactive job can wait for a long time before it gets dispatched. If fast response time is required, interactive jobs should be submitted to high-priority queues with loose scheduling constraints.

Interactive queues

You can configure a queue to be interactive-only, batch-only, or both interactive and batch with the parameter `INTERACTIVE` in `lsb.queues`.

See the *Platform LSF Configuration Reference* for information about configuring interactive queues in the `lsb.queues` file.

Interactive jobs with non-batch utilities

Non-batch utilities such as `lshrun`, `lsgun`, etc., use LIM simple placement advice for host selection when running interactive tasks. For more details on using non-batch utilities to run interactive tasks, see [Running Interactive and Remote Tasks](#) on page 573.

Submitting Interactive Jobs

Use the `bsub -I` option to submit batch interactive jobs, and the `bsub -Is` and `-Ip` options to submit batch interactive jobs in pseudo-terminals.

Pseudo-terminals are not supported for Windows.

For more details, see the `bsub` command.

Finding out which queues accept interactive jobs

Before you submit an interactive job, you need to find out which queues accept interactive jobs with the `bqueues -l` command.

If the output of this command contains the following, this is a batch-only queue. This queue does not accept interactive jobs:

```
SCHEDULING POLICIES: NO_INTERACTIVE
```

If the output contains the following, this is an interactive-only queue:

```
SCHEDULING POLICIES: ONLY_INTERACTIVE
```

If none of the above are defined or if `SCHEDULING POLICIES` is not in the output of `bqueues -l`, both interactive and batch jobs are accepted by the queue.

You configure interactive queues in the `lsb.queues` file.

Submit an interactive job

- 1 Use the `bsub -I` option to submit an interactive batch job.

For example:

```
bsub -I ls
```

Submits a batch interactive job which displays the output of `ls` at the user's terminal.

```
% bsub -I -q interactive -n 4,10 lsmake  
<<Waiting for dispatch ...>>
```

This example starts Platform Make on 4 to 10 processors and displays the output on the terminal.

A new job cannot be submitted until the interactive job is completed or terminated.

When an interactive job is submitted, a message is displayed while the job is awaiting scheduling. The `bsub` command stops display of output from the shell until the job completes, and no mail is sent to the user by default. A user can issue a `ctrl-c` at any time to terminate the job.

Interactive jobs cannot be checkpointed.

Interactive batch jobs cannot be rerunnable (`bsub -r`)

You can submit interactive batch jobs to rerunnable queues (`RERUNNABLE=y` in `lsb.queues`) or rerunnable application profiles (`RERUNNABLE=y` in `lsb.applications`).

Submit an interactive job by using a pseudo-terminal

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either `lshrun` or `bsub` LSF commands.

`bsub -lp`

- 1 To submit a batch interactive job by using a pseudo-terminal, use the `bsub -lp` option.

For example:

```
% bsub -lp vi myfile
```

Submits a batch interactive job to edit `myfile`.

When you specify the `-lp` option, `bsub` submits a batch interactive job and creates a pseudo-terminal when the job starts. Some applications such as `vi` for example, require a pseudo-terminal in order to run correctly.

`bsub -ls`

- 1 To submit a batch interactive job and create a pseudo-terminal with shell mode support, use the `bsub -ls` option.

For example:

```
% bsub -ls csh
```

Submits a batch interactive job that starts up `csh` as an interactive shell.

When you specify the `-ls` option, `bsub` submits a batch interactive job and creates a pseudo-terminal with shell mode support when the job starts. This option should be specified for submitting interactive shells, or applications which redefine the CTRL-C and CTRL-Z keys (for example, `jove`).

Submit an interactive job and redirect streams to files

`bsub -i, -o, -e`

You can use the `-I` option together with the `-i`, `-o`, and `-e` options of `bsub` to selectively redirect streams to files. For more details, see the `bsub(1)` man page.

- 1 To save the standard error stream in the `job.err` file, while standard input and standard output come from the terminal:

```
% bsub -I -q interactive -e job.err lsmake
```

Split stdout and stderr

If in your environment there is a wrapper around `bsub` and LSF commands so that end-users are unaware of LSF and LSF-specific options, you can redirect standard output and standard error of batch interactive jobs to a file with the `>` operator.

By default, both standard error messages and output messages for batch interactive jobs are written to `stdout` on the submission host.

- 1 To write both `stderr` and `stdout` to `mystdout`:

```
bsub -I myjob 2>mystderr 1>mystdout
```

- 2 To redirect both `stdout` and `stderr` to different files, set `LSF_INTERACTIVE_STDERR=y` in `lsf.conf` or as an environment variable.

For example, with `LSF_INTERACTIVE_STDERR` set:

```
bsub -I myjob 2>mystderr 1>mystdout
```

`stderr` is redirected to `mystderr`, and `stdout` to `mystdout`.

See the *Platform LSF Configuration Reference* for more details on `LSF_INTERACTIVE_STDERR`.

Performance Tuning for Interactive Batch Jobs

LSF is often used on systems that support both interactive and batch users. On one hand, users are often concerned that load sharing will overload their workstations and slow down their interactive tasks. On the other hand, some users want to dedicate some machines for critical batch jobs so that they have guaranteed resources. Even if all your workload is batch jobs, you still want to reduce resource contentions and operating system overhead to maximize the use of your resources.

Numerous parameters can be used to control your resource allocation and to avoid undesirable contention.

Types of load conditions

Since interferences are often reflected from the load indices, LSF responds to load changes to avoid or reduce contentions. LSF can take actions on jobs to reduce interference before or after jobs are started. These actions are triggered by different load conditions. Most of the conditions can be configured at both the queue level and at the host level. Conditions defined at the queue level apply to all hosts used by the queue, while conditions defined at the host level apply to all queues using the host.

Scheduling conditions

These conditions, if met, trigger the start of more jobs. The scheduling conditions are defined in terms of load thresholds or resource requirements.

At the queue level, scheduling conditions are configured as either resource requirements or scheduling load thresholds, as described in `lsb.queues`. At the host level, the scheduling conditions are defined as scheduling load thresholds, as described in `lsb.hosts`.

Suspending conditions

These conditions affect running jobs. When these conditions are met, a `SUSPEND` action is performed to a running job.

At the queue level, suspending conditions are defined as `STOP_COND` as described in `lsb.queues` or as suspending load threshold. At the host level, suspending conditions are defined as stop load threshold as described in `lsb.hosts`.

Resuming conditions

These conditions determine when a suspended job can be resumed. When these conditions are met, a `RESUME` action is performed on a suspended job.

At the queue level, resume conditions are defined as by `RESUME_COND` in `lsb.queues`, or by the `loadSched` thresholds for the queue if `RESUME_COND` is not defined.

Types of load indices

To effectively reduce interference between jobs, correct load indices should be used properly. Below are examples of a few frequently used parameters.

Paging rate (pg)

The paging rate (`pg`) load index relates strongly to the perceived interactive performance. If a host is paging applications to disk, the user interface feels very slow.

The paging rate is also a reflection of a shortage of physical memory. When an application is being paged in and out frequently, the system is spending a lot of time performing overhead, resulting in reduced performance.

The paging rate load index can be used as a threshold to either stop sending more jobs to the host, or to suspend an already running batch job to give priority to interactive users.

This parameter can be used in different configuration files to achieve different purposes. By defining paging rate threshold in `lsf.cluster.cluster_name`, the host will become busy from LIM's point of view; therefore, no more jobs will be advised by LIM to run on this host.

By including paging rate in queue or host scheduling conditions, jobs can be prevented from starting on machines with a heavy paging rate, or can be suspended or even killed if they are interfering with the interactive user on the console.

A job suspended due to `pg` threshold will not be resumed even if the resume conditions are met unless the machine is interactively idle for more than `PG_SUSP_IT` seconds.

Interactive idle time (it)

Strict control can be achieved using the idle time (`it`) index. This index measures the number of minutes since any interactive terminal activity. Interactive terminals include hard wired `ttys`, `rlogin` and `lslogin` sessions, and X shell windows such as `xterm`. On some hosts, LIM also detects mouse and keyboard activity.

This index is typically used to prevent batch jobs from interfering with interactive activities. By defining the suspending condition in the queue as `it<1 && pg>50`, a job from this queue will be suspended if the machine is not interactively idle and the paging rate is higher than 50 pages per second. Furthermore, by defining the resuming condition as `it>5 && pg<10` in the queue, a suspended job from the queue will not resume unless it has been idle for at least five minutes and the paging rate is less than ten pages per second.

The `it` index is only non-zero if no interactive users are active. Setting the `it` threshold to five minutes allows a reasonable amount of think time for interactive users, while making the machine available for load sharing, if the users are logged in but absent.

For lower priority batch queues, it is appropriate to set an `it` suspending threshold of two minutes and scheduling threshold of ten minutes in the `lsb.queues` file. Jobs in these queues are suspended while the execution host is in use, and resume after the host has been idle for a longer period. For hosts where all batch jobs, no matter how important, should be suspended, set a per-host suspending threshold in the `lsb.hosts` file.

CPU run queue length (`r15s`, `r1m`, `r15m`)

Running more than one CPU-bound process on a machine (or more than one process per CPU for multiprocessors) can reduce the total throughput because of operating system overhead, as well as interfering with interactive users. Some tasks such as compiling can create more than one CPU-intensive task.

Queues should normally set CPU run queue scheduling thresholds below 1.0, so that hosts already running compute-bound jobs are left alone. LSF scales the run queue thresholds for multiprocessor hosts by using the effective run queue lengths, so multiprocessors automatically run one job per processor in this case.

For concept of effective run queue lengths, see `lsfintro(1)`.

For short to medium-length jobs, the `r1m` index should be used. For longer jobs, you might want to add an `r15m` threshold. An exception to this are high priority queues, where turnaround time is more important than total throughput. For high priority queues, an `r1m` scheduling threshold of 2.0 is appropriate.

CPU utilization (`ut`)

The `ut` parameter measures the amount of CPU time being used. When all the CPU time on a host is in use, there is little to gain from sending another job to that host unless the host is much more powerful than others on the network. A `ut` threshold of 90% prevents jobs from going to a host where the CPU does not have spare processing cycles.

If a host has very high `pg` but low `ut`, then it may be desirable to suspend some jobs to reduce the contention.

Some commands report `ut` percentage as a number from 0-100, some report it as a decimal number between 0-1. The configuration parameter in the `lsf.cluster.cluster_name` file and the configuration files take a fraction in the range from 0 to 1, while the `bsub -R` resource requirement string takes an integer from 1-100.

The command `bhist` shows the execution history of batch jobs, including the time spent waiting in queues or suspended because of system load.

The command `bjobs -p` shows why a job is pending.

Scheduling conditions and resource thresholds

Three parameters, `RES_REQ`, `STOP_COND` and `RESUME_COND`, can be specified in the definition of a queue. Scheduling conditions are a more general way for specifying job dispatching conditions at the queue level. These parameters take resource requirement strings as values which allows you to specify conditions in a more flexible manner than using the `loadSched` or `loadStop` thresholds.

Interactive Batch Job Messaging

LSF can display messages to `stderr` or the Windows console when the following changes occur with interactive batch jobs:

- ◆ Job state
- ◆ Pending reason
- ◆ Suspend reason

Other job status changes, like switching the job's queue, are not displayed.

Limitations

Interactive batch job messaging is not supported in a MultiCluster environment.

Windows

Interactive batch job messaging is not fully supported on Windows. Only changes in the job state that occur before the job starts running are displayed. No messages are displayed after the job starts.

Configure interactive batch job messaging

Messaging for interactive batch jobs can be specified cluster-wide or in the user environment.

Cluster level

- 1 To enable interactive batch job messaging for all users in the cluster, the LSF administrator configures the following parameters in `lsf.conf`:
 - ◆ `LSB_INTERACT_MSG_ENH=Y`
 - ◆ (Optional) `LSB_INTERACT_MSG_INTVAL`

`LSB_INTERACT_MSG_INTVAL` specifies the time interval, in seconds, in which LSF updates messages about any changes to the pending status of the job. The default interval is 60 seconds. `LSB_INTERACT_MSG_INTVAL` is ignored if `LSB_INTERACT_MSG_ENH` is not set.

User level

- 1 To enable messaging for interactive batch jobs, LSF users can define `LSB_INTERACT_MSG_ENH` and `LSB_INTERACT_MSG_INTVAL` as environment variables.

The user-level definition of `LSB_INTERACT_MSG_ENH` overrides the definition in `lsf.conf`.

Example messages

Job in pending state

The following example shows messages displayed when a job is in pending state:

```
bsub -Is -R "ls < 2" csh
Job <2812> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>

<< Job's resource requirements not satisfied: 2 hosts; >>
<< Load information unavailable: 1 host; >>
```

Running X Applications with bsub

```
<< Just started a job recently: 1 host; >>  
<< Load information unavailable: 1 host; >>  
<< Job's resource requirements not satisfied: 1 host; >>
```

Job terminated by user

The following example shows messages displayed when a job in pending state is terminated by the user:

```
bsub -m hostA -b 13:00 -Is sh  
Job <2015> is submitted to default queue <normal>.  
Job will be scheduled after Fri Nov 19 13:00:00 1999  
<<Waiting for dispatch ...>>  
  
<< New job is waiting for scheduling >>  
  
<< The job has a specified start time >>  
  
bkill 2015  
<< Job <2015> has been terminated by user or administrator >>  
  
<<Terminated while pending>>
```

Job suspended then resumed

The following example shows messages displayed when a job is dispatched, suspended, and then resumed:

```
bsub -m hostA -Is sh  
Job <2020> is submitted to default queue <normal>.  
<<Waiting for dispatch ...>>  
  
<< New job is waiting for scheduling >>  
<<Starting on hostA>>  
bstop 2020  
<< The job was suspended by user >>  
  
bresume 2020  
<< Waiting for re-scheduling after being resumed by user >>
```

Running X Applications with bsub

You can start an X session on the least loaded host by submitting it as a batch job:

```
bsub xterm
```

An `xterm` is started on the least loaded host in the cluster.

When you run X applications using `lshrun` or `bsub`, the environment variable `DISPLAY` is handled properly for you. It behaves as if you were running the X application on the local machine.

Writing Job Scripts

You can build a job file one line at a time, or create it from another file, by running `bsub` without specifying a job to submit. When you do this, you start an interactive session in which `bsub` reads command lines from the standard input and submits them as a single batch job. You are prompted with `bsub>` for each line.

You can use the `bsub -Zs` command to spool a file.

For more details on `bsub` options, see the `bsub(1)` man page.

Writing a job file one line at a time

UNIX example

```
% bsub -q simulation
bsub> cd /work/data/myhomedir
bsub> myjob arg1 arg2 .....
bsub> rm myjob.log
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

In the above example, the 3 command lines run as a Bourne shell (`/bin/sh`) script. Only valid Bourne shell command lines are acceptable in this case.

Windows example

```
C:\> bsub -q simulation
bsub> cd \\server\data\myhomedir
bsub> myjob arg1 arg2 .....
bsub> del myjob.log
bsub> ^Z
Job <1234> submitted to queue <simulation>.
```

In the above example, the 3 command lines run as a batch file (`.BAT`). Note that only valid Windows batch file command lines are acceptable in this case.

Specifying job options in a file

In this example, options to run the job are specified in the `options_file`.

```
% bsub -q simulation < options_file
Job <1234> submitted to queue <simulation>.
```

UNIX On UNIX, the `options_file` must be a text file that contains Bourne shell command lines. It cannot be a binary executable file.

Windows On Windows, the `options_file` must be a text file containing Windows batch file command lines.

Spooling a job command file

Use `bsub -Zs` to spool a job command file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, and use the spooled file as the command file for the job.

Use the `bmod -Zsn` command to modify or remove the command file after the job has been submitted. Removing or modifying the original input file does not affect the submitted job.

Redirecting a script to bsub standard input

You can redirect a script to the standard input of the `bsub` command:

```
% bsub < myscript
Job <1234> submitted to queue <test>.
```

In this example, the `myscript` file contains job submission options as well as command lines to execute. When the `bsub` command reads a script from its standard input, it can be modified right after `bsub` returns for the next job submission.

When the script is specified on the `bsub` command line, the script is not spooled:

```
% bsub myscript
Job <1234> submitted to default queue <normal>.
```

In this case the command line `myscript` is spooled, instead of the contents of the `myscript` file. Later modifications to the `myscript` file can affect job behavior.

Specifying embedded submission options

You can specify job submission options in scripts read from standard input by the `bsub` command using lines starting with `#BSUB`:

```
% bsub -q simulation
bsub> #BSUB -q test
bsub> #BSUB -o outfile -R "mem>10"
bsub> myjob arg1 arg2
bsub> #BSUB -J simjob
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

Note that:

- ◆ Command-line options override embedded options. In this example, the job is submitted to the `simulation` queue rather than the `test` queue.
- ◆ Submission options can be specified anywhere in the standard input. In the above example, the `-J` option of `bsub` is specified after the command to be run.
- ◆ More than one option can be specified on one line, as shown in the example above.

Running a job under a particular shell

By default, LSF runs batch jobs using the Bourne (`/bin/sh`) shell. You can specify the shell under which a job is to run. This is done by specifying an interpreter in the first line of the script.

For example:

```
% bsub
bsub> #!/bin/csh -f
bsub> set coredump='ls |grep core'
bsub> if ( "$coredump" != "" ) then
bsub> mv core core.`date | cut -d" " -f1`
bsub> endif
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

The `bsub` command must read the job script from standard input to set the execution shell. If you do not specify a shell in the script, the script is run using `/bin/sh`. If the first line of the script starts with a `#` not immediately followed by an exclamation mark (`!`), then `/bin/csh` is used to run the job.

For example:

```
% bsub
bsub> # This is a comment line. This tells the system to use /bin/csh
to
bsub> # interpret the script.
bsub>
bsub> setenv DAY `date | cut -d" " -f1`
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

If running jobs under a particular shell is required frequently, you can specify an alternate shell using a command-level job starter and run your jobs interactively. See [Controlling Execution Environment Using Job Starters](#) on page 541 for more details.

Registering utmp File Entries for Interactive Batch Jobs

LSF administrators can configure the cluster to track user and account information for interactive batch jobs submitted with `bsub -Ip` or `bsub -Is`. User and account information is registered as entries in the UNIX `utmp` file, which holds information for commands such as `who`. Registering user information for interactive batch jobs in `utmp` allows more accurate job accounting.

Configuration and operation

To enable `utmp` file registration, the LSF administrator sets the `LSB_UTMP` parameter in `lsf.conf`.

When `LSB_UTMP` is defined, LSF registers the job by adding an entry to the `utmp` file on the execution host when the job starts. After the job finishes, LSF removes the entry for the job from the `utmp` file.

Limitations

- ◆ Registration of `utmp` file entries is supported on the following platforms:
 - ❖ SGI IRIX (6.4 and later)
 - ❖ Solaris (all versions)
 - ❖ HP-UX (all versions)
 - ❖ Linux (all versions)
- ◆ `utmp` file registration is not supported in a MultiCluster environment.
- ◆ Because interactive batch jobs submitted with `bsub -I` are not associated with a pseudo-terminal, `utmp` file registration is not supported for these jobs.

Registering utmp File Entries for Interactive Batch Jobs

Running Interactive and Remote Tasks

This chapter provides instructions for running tasks interactively and remotely with non-batch utilities such as `lsrun`, `lsgun`, and `lslogin`.

Contents

- ◆ [Running Remote Tasks](#) on page 573
- ◆ [Interactive Tasks](#) on page 576
- ◆ [Load Sharing Interactive Sessions](#) on page 578
- ◆ [Load Sharing X Applications](#) on page 578

Running Remote Tasks

`lsrun` is a non-batch utility to run tasks on a remote host. `lsgun` is a non-batch utility to run the same task on many hosts, in sequence one after the other, or in parallel.

The default for `lsrun` is to run the job on the host with the least CPU load (represented by the lowest normalized CPU run queue length) and the most available memory. Command-line arguments can be used to select other resource requirements or to specify the execution host.

To avoid typing in the `lsrun` command every time you want to execute a remote job, you can also use a shell alias or script to run your job.

For a complete description of `lsrun` and `lsgun` options, see the `lsrun(1)` and `lsgun(1)` man pages.

In this section

- ◆ [Run a task on the best available host](#) on page 574
- ◆ [Run a task on a host with specific resources](#) on page 574
- ◆ [Run a task on a specific host](#) on page 575
- ◆ [Run a task by using a pseudo-terminal](#) on page 575
- ◆ [Run the same task on many hosts in sequence](#) on page 575
- ◆ [Run parallel tasks](#) on page 575

- ◆ [Run tasks on hosts specified by a file](#) on page 576

Run a task on the best available host

- 1 To run `mytask` on the best available host, enter:

```
lsrun mytask
```

LSF automatically selects a host of the same type as the local host, if one is available. By default the host with the lowest CPU and memory load is selected.

Run a task on a host with specific resources

If you want to run `mytask` on a host that meets specific resource requirements, you can specify the resource requirements using the `-R res_req` option of `lsrun`.

- 1 `lsrun -R 'cserver && swp>100' mytask`

In this example `mytask` must be run on a host that has the resource `cserver` and at least 100 MB of virtual memory available.

You can also configure LSF to store the resource requirements of specific tasks. If you configure LSF with the resource requirements of your task, you do not need to specify the `-R res_req` option of `lsrun` on the command-line. If you do specify resource requirements on the command line, they override the configured resource requirements.

See the *Platform LSF Configuration Reference* for information about configuring resource requirements in the `lsf.task` file.

Resource usage

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]: ... :res[=value]
```

The `res` is one of the resources whose value is returned by the `lsload` command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

Run a task on a specific host

- 1 If you want to run your task on a particular host, use the `lsrun -m` option:

```
lsrun -m hostD mytask
```

Run a task by using a pseudo-terminal

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either `lsrun` or `bsub` LSF commands.

Some tasks, such as text editors, require special terminal handling. These tasks must be run using a pseudo-terminal so that special terminal handling can be used over the network.

- 1 The `-P` option of `lsrun` specifies that the job should be run using a pseudo-terminal:

```
lsrun -P vi
```

Run the same task on many hosts in sequence

The `lsgrun` command allows you to run the same task on many hosts, one after the other, or in parallel.

- 1 For example, to merge the `/tmp/out` file on hosts `hostA`, `hostD`, and `hostB` into a single file named `gout`, enter:

```
lsgrun -m "hostA hostD hostB" cat /tmp/out >> gout
```

Run parallel tasks

`lsgrun -p`

The `-p` option tells `lsgrun` that the task specified should be run in parallel. See `lsgrun(1)` for more details.

- 1 To remove the `/tmp/core` file from all 3 hosts, enter:

```
lsgrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

Run tasks on hosts specified by a file

`lsgrun -f host_file` 1 The `lsgrun -f host_file` option reads the *host_file* file to get a list of hosts on which to run the task.

Interactive Tasks

LSF supports transparent execution of tasks on all server hosts in the cluster. You can run your program on the best available host and interact with it just as if it were running directly on your workstation. Keyboard signals such as CTRL-Z and CTRL-C work as expected.

Interactive tasks communicate with the user in real time. Programs like `vi` use a text-based terminal interface. Computer Aided Design and desktop publishing applications usually use a graphic user interface (GUI).

This section outlines issues for running interactive tasks with the non-batch utilities `lsrun`, `lsgrun`, etc. To run interactive tasks with these utilities, use the `-i` option.

For more details, see the `lsrun(1)` and `lsgrun(1)` man pages.

In this section

- ◆ [Interactive tasks on remote hosts](#) on page 576
- ◆ [Interactive processing and scheduling policies](#) on page 577
- ◆ [Shared files and user IDs](#) on page 577
- ◆ [Shell mode for remote execution](#) on page 577
- ◆ [Run windows](#) on page 577
- ◆ [Redirect streams to files](#) on page 577

Interactive tasks on remote hosts

Job controls

When you run an interactive task on a remote host, you can perform most of the job controls as if it were running locally. If your shell supports job control, you can suspend and resume the task and bring the task to background or foreground as if it were a local task.

For a complete description, see the `lsrun(1)` man page.

Hiding remote execution

You can also write one-line shell scripts or `csh` aliases to hide remote execution. For example:

```
#!/bin/sh
# Script to remotely execute mytask
exec lsrun -m hostD mytask

or

alias mytask "lsrun -m hostD mytask"
```


Interactive processing and scheduling policies

LSF lets you run interactive tasks on any computer on the network, using your own terminal or workstation. Interactive tasks run immediately and normally require some input through a text-based or graphical user interface. All the input and output is transparently sent between the local host and the job execution host.

Shared files and user IDs

When LSF runs a task on a remote host, the task uses standard UNIX system calls to access files and devices. The user must have an account on the remote host. All operations on the remote host are done with the user's access permissions.

Tasks that read and write files access the files on the remote host. For load sharing to be transparent, your files should be available on all hosts in the cluster using a file sharing mechanism such as NFS or AFS. When your files are available on all hosts in the cluster, you can run your tasks on any host without worrying about how your task will access files.

LSF can operate correctly in cases where these conditions are not met, but the results may not be what you expect. For example, the `/tmp` directory is usually private on each host. If you copy a file into `/tmp` on a remote host, you can only read that file on the same remote host.

LSF can also be used when files are not available on all hosts. LSF provides the `lsrscp` command to copy files across LSF hosts. You can use pipes to redirect the standard input and output of remote commands, or write scripts to copy the data files to the execution host.

Shell mode for remote execution

On UNIX, shell mode support is provided for running interactive applications through RES.

Not supported for Windows.

Shell mode support is required for running interactive shells or applications that redefine the `CTRL-C` and `CTRL-Z` keys (for example, `jove`).

The `-S` option of `lsrun`, `ch` or `lsgrun` creates the remote task with shell mode support. The default is not to enable shell mode support.

Run windows

Some run windows are only applicable to batch jobs. Interactive jobs scheduled by LIM are controlled by another set of run windows.

Redirect streams to files

By default, both standard error messages and standard output messages of interactive tasks are written to `stdout` on the submission host.

To separate `stdout` and `stderr` and redirect to separate files, set `LSF_INTERACTIVE_STDERR=y` in `lsf.conf` or as an environment variable.

-
- 1 To redirect both `stdout` and `stderr` to different files with the parameter set:


```
lsrun mytask 2>mystderr 1>mystdout
```

The result of the above example is for `stderr` to be redirected to `mystderr`, and `stdout` to `mystdout`. Without `LSF_INTERACTIVE_STDERR` set, both `stderr` and `stdout` will be redirected to `mystdout`.

See the *Platform LSF Configuration Reference* for more details on `LSF_INTERACTIVE_STDERR`.

Load Sharing Interactive Sessions

There are different ways to use LSF to start an interactive session on the best available host.

Log on to the least loaded host

- 1 To log on to the least loaded host, use the `lslogin` command.

When you use `lslogin`, LSF automatically chooses the best host and does an `rlogin` to that host.

With no argument, `lslogin` picks a host that is lightly loaded in CPU, has few login sessions, and whose binary is compatible with the current host.

Log on to a host with specific resources

- 1 If you want to log on a host that meets specific resource requirements, use the `lslogin -R res_req` option.

```
lslogin -R "solaris order[ls:cpu]"
```

This command opens a remote login to a host that has the `sunos` resource, few other users logged in, and a low CPU load level. This is equivalent to using `lsplace` to find the best host and then using `rlogin` to log in to that host:

```
rlogin `lsplace -R "sunos order[ls:cpu]"`
```

Load Sharing X Applications

Start an xterm

- 1 If you are using the X Window System, you can start an `xterm` that opens a shell session on the least loaded host by entering:

```
lsrun sh -c xterm &
```

The `&` in this command line is important as it frees resources on the host once `xterm` is running, by running the X terminal in the background.

In this example, no processes are left running on the local host. The `lsrun` command exits as soon as `xterm` starts, and the `xterm` on the remote host connects directly to the X server on the local host.

xterm on a PC

Each X application makes a separate network connection to the X display on the user's desktop. The application generally gets the information about the display from the DISPLAY environment variable.

X-based systems such as `eXceed` start applications by making a remote shell connection to the UNIX server, setting the DISPLAY environment variable, and then invoking the X application. Once the application starts, it makes its own connection to the display and the initial remote shell is no longer needed.

This approach can be extended to allow load sharing of remote applications. The client software running on the X display host makes a remote shell connection to any server host in the LSF cluster. Instead of running the X application directly, the client invokes a script that uses LSF to select the best available host and starts the application on that host. Because the application then makes a direct connection to the display, all of the intermediate connections can be closed. The client software on the display host must select a host in the cluster to start the connection. You can choose an arbitrary host for this; once LSF selects the best host and starts the X application there, the initial host is no longer involved. There is no ongoing load on the initial host.

Setting up an X terminal to start an X session on the least loaded host

If you are using a PC as a desktop machine and are running an X Window server on your PC, then you can start an X session on the least loaded host.

The following steps assume you are using `Exceed` from Hummingbird Communications. This procedure can be used to load share any X-based application.

You can customize host selection by changing the resource requirements specified with `-R ". . . "`. For example, a user could have several icons in the `xterm` program group: one called `Best`, another called `Best_Sun`, another `Best_SGI`.

Set up Exceed to log on the least loaded host

To set up `Exceed` to log on to the least loaded host:

- 1 Click the Xstart icon in the Exceed program group.
- 2 Choose REXEC (TCP/IP, ...) as start method, program type is X window.
- 3 Set the host to be any server host in your LSF cluster:
- 4 Set description to be `Best`.
- 5 Click the Install button in the Xstart window.

This installs `Best` as an icon in the program group you chose (for example, `xterm`).

The user can now log on to the best host by clicking `Best` in the Xterm program group.

```
lslrun -R "type==any order[cpu:mem:login]" xterm -sb -ls -display your_PC:0.0
```

Start an xterm in Exceed

To start an xterm:

- 1 Double-click the Best icon.

An xterm starts on the least loaded host in the cluster and is displayed on your screen.

Examples

Running any application on the least loaded host

To run appY on the best machine licensed for it, you could set the command line in Exceed to be the following and set the description to appY:

```
lsrun -R "type==any && appY order[mem:cpu]" sh -c "appY -display your_PC:0.0 &"
```

You must make sure that all the UNIX servers licensed for appY are configured with the resource "appY". In this example, appY requires a lot of memory when there are embedded graphics, so we make "mem" the most important consideration in selecting the best host among the eligible servers.

Starting an X session on the least loaded host in any X desktop environment

The above approach also applies to other X desktop environments. In general, if you want to start an X session on the best host, run the following on an LSF host:

```
lsrun -R "resource_requirement" my_Xapp -display your_PC:0.0
```

where

resource_requirement is your resource requirement string

Script for automatically specifying resource requirements

The above examples require the specification of resource requirement strings by users. You may want to centralize this such that all users use the same resource specifications.

You can create a central script (for example `lslaunch`) and place it in the `/lsf/bin` directory. For example:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" $@
exit $?
```

Which would simplify the command string to:

```
lslaunch xterm -sb -ls -display your_PC:0.0
```

Taking this one step further, you could create a script named `lsxterm`:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" xterm -sb -ls $@
exit $?
```

Which would simplify the command string to:

```
lsxterm -display your_PC:0.0
```

VII

Monitoring Your Cluster

- ◆ [Achieving Performance and Scalability](#) on page 583
- ◆ [Reporting](#) on page 597
- ◆ [Event Generation](#) on page 623
- ◆ [Tuning the Cluster](#) on page 627
- ◆ [Authentication and Authorization](#) on page 641
- ◆ [Job Email and Job File Spooling](#) on page 649
- ◆ [Non-Shared File Systems](#) on page 655
- ◆ [Error and Event Logging](#) on page 661
- ◆ [Troubleshooting and Error Messages](#) on page 671
- ◆ [Understanding Platform LSF Job Exit Information](#) on page 693

Achieving Performance and Scalability

Contents

- ◆ [Optimizing Performance in Large Sites](#) on page 583
- ◆ [Tuning UNIX for Large Clusters](#) on page 584
- ◆ [Tuning LSF for Large Clusters](#) on page 585
- ◆ [Monitoring Performance Metrics in Real Time](#) on page 593

Optimizing Performance in Large Sites

As your site grows, you must tune your LSF cluster to support a large number of hosts and an increased workload.

This chapter discusses how to efficiently tune querying, scheduling, and event logging in a large cluster that scales to 5000 hosts and 100,000 jobs at any one time.

To target performance optimization to a cluster with 5000 hosts and 100,000 jobs, you must:

- ◆ Configure your operating system. See [Tuning UNIX for Large Clusters](#) on page 584
- ◆ Fine-tune LSF. See [Tuning LSF for Large Clusters](#) on page 585

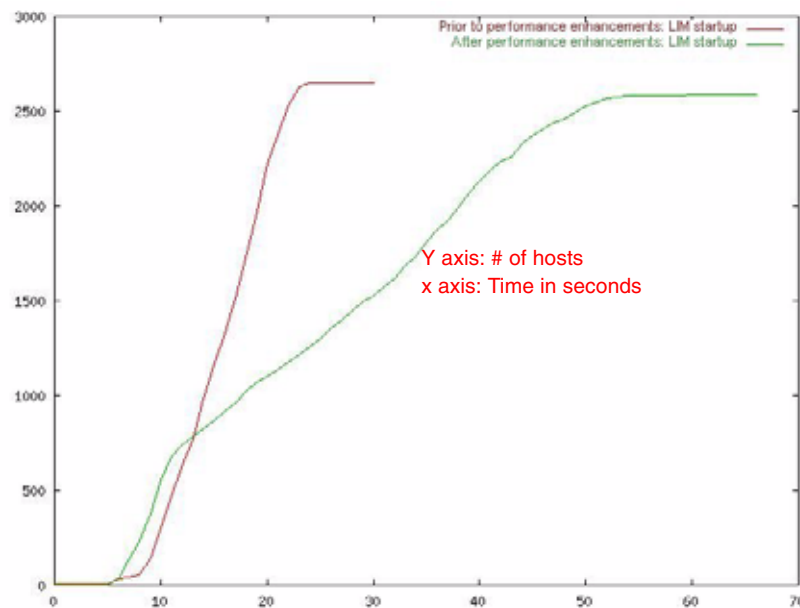
What's new in LSF performance?

LSF provides parameters for tuning your cluster, which you will learn about in this chapter. However, before you calculate the values to use for tuning your cluster, consider the following enhancements to the general performance of LSF daemons, job dispatching, and event replaying:

- ◆ Both scheduling and querying are much faster
- ◆ Switching and replaying the events log file, `lsb.events`, is much faster. The length of the events file no longer impacts performance
- ◆ Restarting and reconfiguring your cluster is much faster
- ◆ Job submission time is constant. It does not matter how many jobs are in the system. The submission time does not vary.
- ◆ The scalability of load updates from the slaves to the master has increased

- ◆ Load update intervals are scaled automatically

The following graph shows the improvement in LIM startup after the LSF performance enhancements:



Tuning UNIX for Large Clusters

The following hardware and software specifications are requirements for a large cluster that supports 5,000 hosts and 100,000 jobs at any one time.

In this section

- ◆ [Hardware recommendation](#) on page 584
- ◆ [Software requirement](#) on page 584

Hardware recommendation

LSF master host:

- ◆ 4 processors, one each for:
 - ◆ mbatchd
 - ◆ mbschd
 - ◆ lim
 - ◆ Operating system
- ◆ 10 GB Ram

Software requirement

To meet the performance requirements of a large cluster, increase the file descriptor limit of the operating system.

The file descriptor limit of most operating systems used to be fixed, with a limit of 1024 open files. Some operating systems, such as Linux and AIX, have removed this limit, allowing you to increase the number of file descriptors.

Increase the file descriptor limit

- 1 To achieve efficiency of performance in LSF, follow the instructions in your operating system documentation to increase the number of file descriptors on the LSF master host.

TIP: To optimize your configuration, set your file descriptor limit to a value at least as high as the number of hosts in your cluster.

The following is an example configuration. The instructions for different operating systems, kernels, and shells are varied. You may have already configured the host to use the maximum number of file descriptors that are allowed by the operating system. On some operating systems, the limit is configured dynamically.

Your cluster size is 5000 hosts. Your master host is on Linux, kernel version 2.4:

- 1 Log in to the LSF master host as the `root` user.
- 2 Add the following line to your `/etc/rc.d/rc.local` startup script:

```
echo -n "5120" > /proc/sys/fs/file-max
```
- 3 Restart the operating system to apply the changes.
- 4 In the `bash` shell, instruct the operating system to use the new file limits:

```
# ulimit -n unlimited
```

Tuning LSF for Large Clusters

To enable and sustain large clusters, you need to tune LSF for efficient querying, dispatching, and event log management.

In this section

- ◆ [Managing scheduling performance](#) on page 585
- ◆ [Limiting the number of batch queries](#) on page 587
- ◆ [Improving the speed of host status updates](#) on page 587
- ◆ [Managing your user's ability to move jobs in a queue](#) on page 588
- ◆ [Managing the number of pending reasons](#) on page 588
- ◆ [Achieving efficient event switching](#) on page 588
- ◆ [Automatic load updating](#) on page 589
- ◆ [Managing the I/O performance of the info directory](#) on page 589
- ◆ [Processor binding for LSF job processes](#) on page 590
- ◆ [Increasing the job ID limit](#) on page 592

Managing scheduling performance

For fast job dispatching in a large cluster, configure the following parameters:

`LSB_MAX_JOB_DISPATCH_PER_SESSION` in `lsf.conf`

The maximum number of jobs the scheduler can dispatch in one scheduling session

Some operating systems, such as Linux and AIX, let you increase the number of file descriptors that can be allocated on the master host. You do not need to limit the number of file descriptors to 1024 if you want fast job dispatching. To take advantage of the greater number of file descriptors, you must set `LSB_MAX_JOB_DISPATCH_PER_SESSION` to a value greater than 300.

Set `LSB_MAX_JOB_DISPATCH_PER_SESSION` to one-half the value of `MAX_SBD_CONNS`. This setting configures `mbatchd` to dispatch jobs at a high rate while maintaining the processing speed of other `mbatchd` tasks.

`MAX_SBD_CONNS` in `lsb.params`

The maximum number of open file connections between `mbatch` and `sbatchd`.

Specify a value equal to the number of hosts in your cluster plus a buffer. For example, if your cluster includes 4000 hosts, set:

```
MAX_SBD_CONNS=4100
```

Highly recommended for large clusters to decrease the load on the master LIM. Forces the client `sbatchd` to contact the local LIM for host status and load information. The client `sbatchd` only contacts the master LIM or a LIM on one of the `LSF_SERVER_HOSTS` if `sbatchd` cannot find the information locally.

Enable fast job dispatch

- 1 Log in to the LSF master host as the `root` user.
- 2 Increase the system-wide file descriptor limit of your operating system if you have not already done so.
- 3 In `lsb.params`, set `MAX_SBD_CONNS` equal to the number of hosts in the cluster plus a buffer.
- 4 In `lsf.conf`, set the parameter `LSB_MAX_JOB_DISPATCH_PER_SESSION` to a value greater than 300 and less than or equal to one-half the value of `MAX_SBD_CONNS`.

For example, for a cluster with 4000 hosts:

```
LSB_MAX_JOB_DISPATCH_PER_SESSION = 2050  
MAX_SBD_CONNS=4100
```

- 5 In `lsf.conf`, define the parameter `LSF_SERVER_HOSTS` to decrease the load on the master LIM.
- 6 In the shell you used to increase the file descriptor limit, shut down the LSF batch daemons on the master host:

```
badadmin hshutdown
```
- 7 Run `badadmin mbdrestart` to restart the LSF batch daemons on the master host.
- 8 Run `badadmin hrestart all` to restart every `sbatchd` in the cluster:

NOTE: When you shut down the batch daemons on the master host, all LSF services are temporarily unavailable, but existing jobs are not affected. When `mbatchd` is later started by `sbatchd`, its previous status is restored and job scheduling continues.

Enable continuous scheduling

- 1 To enable the scheduler to run continuously, define the parameter `JOB_SCHEDULING_INTERVAL=0` in `lsb.params`.

Limiting the number of batch queries

In large clusters, job querying can grow very quickly. If your site sees a lot of high traffic job querying, you can tune LSF to limit the number of job queries that `mbatchd` can handle. This helps decrease the load on the master host.

If a job information query is sent after the limit has been reached, an error message is displayed and `mbatchd` keeps retrying, in one second intervals. If the number of job queries later drops below the limit, `mbatchd` handles the query.

You define the maximum number of concurrent jobs queries to be handled by `mbatchd` in the parameter `MAX_CONCURRENT_JOB_QUERY` in `lsb.params`:

- ◆ If `mbatchd` is using multithreading, a dedicated query port is defined by the parameter `LSB_QUERY_PORT` in `lsf.conf`. When `mbatchd` has a dedicated query port, the value of `MAX_CONCURRENT_JOB_QUERY` sets the maximum number of queries that can be handled by each child `mbatchd` that is forked by `mbatchd`. This means that the total number of job queries handled can be more than the number specified by `MAX_CONCURRENT_JOB_QUERY` (`MAX_CONCURRENT_JOB_QUERY` multiplied by the number of child daemons forked by `mbatchd`).
- ◆ If `mbatchd` is not using multithreading, the value of `MAX_CONCURRENT_JOB_QUERY` sets the maximum total number of job queries that can be handled by `mbatchd`.

Syntax

```
MAX_CONCURRENT_JOB_QUERY=max_query
```

Where:

max_query

Specifies the maximum number of job queries that can be handled by `mbatchd`. Valid values are positive integers between 1 and 100. The default value is unlimited.

Examples

```
MAX_CONCURRENT_JOB_QUERY=20
```

Specifies that no more than 20 queries can be handled by `mbatchd`.

```
MAX_CONCURRENT_JOB_QUERY=101
```

Incorrect value. The default value will be used. An unlimited number of job queries will be handled by `mbatchd`.

Improving the speed of host status updates

To improve the speed with which `mbatchd` obtains and reports host status, configure the parameter `LSB_SYNC_HOST_STAT_LIM` in the file `lsb.params`. This also improves the speed with which LSF reschedules jobs: the sooner LSF knows that a host has become unavailable, the sooner LSF reschedules any rerunnable jobs executing on that host.

For example, during maintenance operations, the cluster administrator might need to shut down half of the hosts at once. LSF can quickly update the host status and reschedule any rerunnable jobs that were running on the unavailable hosts.

When you define this parameter, `mbatchd` periodically obtains the host status from the master LIM, and then verifies the status by polling each `sbatchd` at an interval defined by the parameters `MBD_SLEEP_TIME` and `LSB_MAX_PROBE_SBD`.

Managing your user's ability to move jobs in a queue

`JOB_POSITION_CONTROL_BY_ADMIN=Y` allows an LSF administrator to control whether users can use `btop` and `bbot` to move jobs to the top and bottom of queues. When set, only the LSF administrator (including any queue administrators) can use `bbot` and `btop` to move jobs within a queue. A user attempting to use `bbot` or `btop` receives the error "User permission denied."

REMEMBER: You must be an LSF administrator to set this parameter.

Managing the number of pending reasons

For efficient, scalable management of pending reasons, use `CONDENSE_PENDING_REASONS=Y` in `lsb.params` to condense all the host-based pending reasons into one generic pending reason.

If a job has no other main pending reason, `bjobs -p` or `bjobs -l` will display the following:

```
Individual host based reasons
```

If you condense host-based pending reasons, but require a full pending reason list, you can run the following command:

```
badmin diagnose <job_ID>
```

REMEMBER: You must be an LSF administrator or a queue administrator to run this command.

Achieving efficient event switching

Periodic switching of the event file can weaken the performance of `mbatchd`, which automatically backs up and rewrites the events file after every 1000 batch job completions. The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`.

Change the frequency of event switching with the following two parameters in `lsb.params`:

- ◆ `MAX_JOB_NUM` specifies the number of batch jobs to complete before `lsb.events` is backed up and moved to `lsb.events.1`. The default value is 1000
- ◆ `MIN_SWITCH_PERIOD` controls how frequently `mbatchd` checks the number of completed batch jobs

The two parameters work together. Specify the `MIN_SWITCH_PERIOD` value in seconds.

For example:

```
MAX_JOB_NUM=1000
```

```
MIN_SWITCH_PERIOD=7200
```

This instructs `mbatchd` to check if the events file has logged 1000 batch job completions every two hours. The two parameters can control the frequency of the events file switching as follows:

- ◆ After two hours, `mbatchd` checks the number of completed batch jobs. If 1000 completed jobs have been logged, it switches the events file
- ◆ If 1000 jobs complete after five minutes, `mbatchd` does not switch the events file until till the end of the two-hour period

TIP: For large clusters, set the `MIN_SWITCH_PERIOD` to a value equal to or greater than 600. This causes `mbatchd` to fork a child process that handles event switching, thereby reducing the load on `mbatchd`. `mbatchd` terminates the child process and appends delta events to new events after the `MIN_SWITCH_PERIOD` has elapsed. If you define a value less than 600 seconds, `mbatchd` will not fork a child process for event switching.

Automatic load updating

Periodically, the LIM daemons exchange load information. In large clusters, let LSF automatically load the information by dynamically adjusting the period based on the load.

IMPORTANT: For automatic tuning of the loading interval, make sure the parameter `EXINTERVAL` in `lsf.cluster.cluster_name` file is *not* defined. Do not configure your cluster to load the information at specific intervals.

Managing the I/O performance of the info directory

In large clusters, there are large numbers of jobs submitted by its users. Since each job generally has a job file, this results in a large number of job files stored in the `LSF_SHARED_DIR/cluster_name/logdir/info` directory at any time. When the total size of the job files reaches a certain point, you will notice a significant delay when performing I/O operations in the `info` directory.

This delay is caused by a limit in the total size of files that can reside in a file server directory. This limit is dependent on the file system implementation. A high load on the file server delays the master batch daemon operations, and therefore slows down the overall cluster throughput.

You can prevent this delay by creating and using subdirectories under the parent directory. Each new subdirectory is subject to the file size limit, but the parent directory is not subject to the total file size of its subdirectories. Since the total file size of the `info` directory is divided among its subdirectories, your cluster can process more job operations before reaching the total size limit of the job files.

If your cluster has a lot of jobs resulting in a large `info` directory, you can tune your cluster by enabling LSF to create subdirectories in the `info` directory. Use `MAX_INFO_DIRS` in `lsb.params` to create the subdirectories and enable `mbatchd` to distribute the job files evenly throughout the subdirectories.

Syntax

```
MAX_INFO_DIRS=num_subdirs
```

Where `num_subdirs` specifies the number of subdirectories that you want to create under the `LSF_SHARED_DIR/cluster_name/logdir/info` directory. Valid values are positive integers between 1 and 1024. By default, `MAX_INFO_DIRS` is not defined.

Duplicate event logging

Run `badmin reconfig` to create and use the subdirectories.

NOTE: If you enabled duplicate event logging, you must run `badmin mbdrestart` instead of `badming reconfig` to restart `mbatchd`.

Run `bparams -l` to display the value of the `MAX_INFO_DIRS` parameter.

Example

```
MAX_INFO_DIRS=10
mbatchd creates ten subdirectories from
LSB_SHAREDIR/cluster_name/logdir/info/0 to
LSB_SHAREDIR/cluster_name/logdir/info/9.
```

Processor binding for LSF job processes

Rapid progress of modern processor manufacture technologies has enabled the low cost deployment of LSF on hosts with multicore and multithread processors. The default soft affinity policy enforced by the operating system scheduler may not give optimal job performance. For example, the operating system scheduler may place all job processes on the same processor or core leading to poor performance. Frequently switching processes as the operating system schedules and reschedules work between cores can cause cache invalidations and cache miss rates to grow large.

Processor binding for LSF job processes takes advantage of the power of multiple processors and multiple cores to provide hard processor binding functionality for sequential LSF jobs.

RESTRICTION: Processor binding is supported on hosts running Linux with kernel version 2.6 or higher.

For parallel jobs, LSF binds the job at the first execution host, not other remote hosts. LSF does not bind remote tasks for parallel jobs because parallel jobs typically exclusively allocate entire nodes to a job.

When processor binding for LSF job processes is enabled on supported hosts, job processes of an LSF job are bound to a processor according to the binding policy of the host. When an LSF job is completed (exited or done successfully) or suspended, the corresponding processes are unbound from the processor.

When a suspended LSF job is resumed, the corresponding processes are bound again to a processor. The process is not guaranteed to be bound to the same processor it was bound to before the job was suspended.

The processor binding affects the whole job process group. All job processes forked from the root job process (the job RES) are bound to the same processor.

Processor binding for LSF job processes does not bind daemon processes.

If processor binding is enabled, but the execution hosts do not support processor affinity, the configuration has no effect on the running processes. Processor binding has no effect on a single-processor host.

Processor, core, and thread CPU load balancing

By default, the number of CPUs on a host represents the number of physical processors a machine has. For LSF hosts with multiple cores, threads, and processors, `ncpus` can be defined by the cluster administrator to consider one of the following:

- ◆ Processors
 - Processors and cores
 - Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for `ncpus` is to consider only the number of physical processors (`EGO_DEFINE_NCPUS=procs`).

Binding job processes randomly to multiple processors, cores, or threads, may affect job performance. Processor binding configured with `LSF_BIND_JOB` in `lsf.conf` or `BIND_JOB` in `lsb.applications`, detects the `EGO_DEFINE_NCPUS` policy to load balance the job processes by processor, core, or thread (PCT).

For example, if a host's PCT policy is set to processor (`EGO_DEFINE_NCPUS=procs`), the first job process is bound to the first physical processor, the second job process is bound to the second physical processor and so on.

If host's PCT policy is set to core level (`EGO_DEFINE_NCPUS=cores`), the first job process is bound to the first core on the first physical processor, the second job process is bound to the first core on the second physical processor, the third job process is bound to the second core on the first physical processor and so on.

If host's PCT policy is set to thread level (`EGO_DEFINE_NCPUS=threads`), the first job process is bound to the first thread on the first physical processor, the second job process is bound to the first thread on the second physical processor, the third job process is bound to the second thread on the first physical processor and so on.

Feature Interactions

- ◆ Existing CPU affinity features
 - Processor binding of LSF job processes will not take effect on a master host with the following parameters configured.
 - ❖ `MBD_QUERY_CPUS`
 - ❖ `LSF_DAEMONS_CPUS`
 - ❖ `EGO_DAEMONS_CPUS`
- ◆ IRIX cpusets
 - Processor binding cannot be used with IRIX cpusets. If an execution host is configured as part of a cuset, processor binding is disabled on that host.
- ◆ Job requeue, rerun, and migration
 - When a job is requeued, rerun or migrated, a new job process is created. If processor binding is enabled when the job runs, the job processes will be bound to a processor.
- ◆ `badmin hrestart`

`badadmin hrestart` restarts a new `sbatchd`. If a job process has already been bound to a processor, after `sbatchd` is restarted, processor binding for the job processes are restored.

- ◆ `badadmin reconfig`

If the `BIND_JOB` parameter is modified in an application profile, `badadmin reconfig` only affects pending jobs. The change does not affect running jobs.

- ◆ MultiCluster job forwarding model

In a MultiCluster environment, the behavior is similar to the current application profile behavior. If the application profile name specified in the submission cluster is not defined in the execution cluster, the job is rejected. If the execution cluster has the same application profile name, but does not enable processor binding, the job processes are not bound at the execution cluster.

Enable processor binding for LSF job processes

- 1 Enable processor binding cluster-wide or in an application profile.

- ❖ Cluster-wide configuration (`lsf.conf`)

Use `LSF_BIND_JOB=Y` in `lsf.conf` to enable processor binding for all execution hosts in the cluster. On the execution hosts that support this feature, job processes will be hard bound to selected processors.

- ❖ Application profile configuration (`lsb.applications`)

Use `BIND_JOB=Y` in an application profile configuration in `lsb.applications` to enable processor binding for all jobs submitted to the application profile. On the execution hosts that support this feature, job processes will be hard bound to selected processors.

If `BIND_JOB` is not set in an application profile in `lsb.applications`, the value of `LSF_BIND_JOB` in `lsf.conf` takes effect. The `BIND_JOB` parameter configured in an application profile overrides the `lsf.conf` setting.

Increasing the job ID limit

By default, LSF assigns job IDs up to 6 digits. This means that no more than 999999 jobs can be in the system at once. The job ID limit is the highest job ID that LSF will ever assign, and also the maximum number of jobs in the system.

LSF assigns job IDs in sequence. When the job ID limit is reached, the count rolls over, so the next job submitted gets job ID "1". If the original job 1 remains in the system, LSF skips that number and assigns job ID "2", or the next available job ID. If you have so many jobs in the system that the low job IDs are still in use when the maximum job ID is assigned, jobs with sequential numbers could have different submission times.

Increase the maximum job ID

You cannot lower the job ID limit, but you can raise it to 10 digits. This allows longer term job accounting and analysis, and means you can have more jobs in the system, and the job ID numbers will roll over less often.

Use `MAX_JOBID` in `lsb.params` to specify any integer from 999999 to 2147483646 (for practical purposes, you can use any 10-digit integer less than this value).

Increase the job ID display length

By default, `bjobs` and `bhist` display job IDs with a maximum length of 7 characters. Job IDs greater than 9999999 are truncated on the left.

Use `LSB_JOBID_DISP_LENGTH` in `lsf.conf` to increase the width of the JOBID column in `bjobs` and `bhist` display. When `LSB_JOBID_DISP_LENGTH=10`, the width of the JOBID column in `bjobs` and `bhist` increases to 10 characters.

Monitoring Performance Metrics in Real Time

Enable metric collection

Set `SCHED_METRIC_ENABLE=Y` in `lsb.params` to enable performance metric collection.

Start performance metric collection dynamically:

```
badmin perfmon start sample_period
```

Optionally, you can set a sampling period, in seconds. If no sample period is specified, the default sample period set in `SCHED_METRIC_SAMPLE_PERIOD` in `lsb.params` is used.

Stop sampling:

```
badmin perfmon stop
```

`SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` can be specified independently. That is, you can specify `SCHED_METRIC_SAMPLE_PERIOD` and not specify `SCHED_METRIC_ENABLE`. In this case, when you turn on the feature dynamically (using `badmin perfmon start`), the sampling period valued defined in `SCHED_METRIC_SAMPLE_PERIOD` will be used.

`badmin perfmon start` and `badmin perfmon stop` override the configuration setting in `lsb.params`. Even if `SCHED_METRIC_ENABLE` is set, if you run `badmin perfmon start`, performance metric collection is started. If you run `badmin perfmon stop`, performance metric collection is stopped.

Tune the metric sampling period

Set `SCHED_METRIC_SAMPLE_PERIOD` in `lsb.params` to specify an initial cluster-wide performance metric sampling period.

Set a new sampling period in seconds:

```
badmin perfmon setperiod sample_period
```

Collecting and recording performance metric data may affect the performance of LSF. Smaller sampling periods will result in the `lsb.streams` file growing faster.

Display current performance

Run `badmin perfmon view` to view real time performance metric information. The following metrics are collected and recorded in each sample period:

- ◆ The number of queries handled by `mbatchd`
- ◆ The number of queries for each of jobs, queues, and hosts. (`bjobs`, `bqueues`, and `bhosts`, as well as other daemon requests)
- ◆ The number of jobs submitted (divided into job submission requests and jobs actually submitted)

Monitoring Performance Metrics in Real Time

- ◆ The number of jobs dispatched
- ◆ The number of jobs completed
- ◆ The numbers of jobs sent to remote cluster
- ◆ The numbers of jobs accepted by from cluster

`badmin perfmon view`

Performance monitor start time: Fri Jan 19 15:07:54

End time of last sample period: Fri Jan 19 15:25:55

Sample period : 60 Seconds

Metrics	Last	Max	Min	Avg	Total
Total queries	0	25	0	8	159
Jobs information queries	0	13	0	2	46
Hosts information queries	0	0	0	0	0
Queue information queries	0	0	0	0	0
Job submission requests	0	10	0	0	10
Jobs submitted	0	100	0	5	100
Jobs dispatched	0	0	0	0	0
Jobs completed	0	13	0	5	100
Jobs sent to remote cluster	0	12	0	5	100
Jobs accepted from remote cluster	0	0	0	0	0

Performance metrics information is calculated at the end of each sampling period. Running `badmin perfmon` before the end of the sampling period displays metric data collected from the sampling start time to the end of last sample period.

If no metrics have been collected because the first sampling period has not yet ended, `badmin perfmon view` displays:

`badmin perfmon view`

Performance monitor start time: Thu Jan 25 22:11:12

End time of last sample period: Thu Jan 25 22:11:12

Sample period : 120 Seconds

No performance metric data available. Please wait until first sample period ends.

badmin perfmon output

Sample Period	Current sample period
Performance monitor start time	The start time of sampling
End time of last sample period	The end time of last sampling period
Metric	The name of metrics
Total	This is accumulated metric counter value for each metric. It is counted from Performance monitor start time to End time of last sample period.

Last Period Last sampling value of metric. It is calculated per sampling period. It is represented as the metric value per period, and normalized by the following formula.

$$\text{LastPeriod} = \frac{\text{Metric Counter Value of Last Period}}{\text{Sample Period Interval}} \times \text{Sample Period}$$

Max Maximum sampling value of metric. It is re-evaluated in each sampling period by comparing Max and Last Period. It is represented as the metric value per period.

Min Minimum sampling value of metric. It is re-evaluated in each sampling period by comparing Min and Last Period. It is represented as the metric value per period.

Avg Average sampling value of metric. It is recalculated in each sampling period. It is represented as the metric value per period, and normalized by the following formula.

$$\text{Avg} = \frac{\text{Total}}{\text{LastPeriodEndTime} - \text{SampleStartTime}} \times \text{Sample Period}$$

Reconfiguring your cluster with performance metric sampling enabled

badadmin mbdrestart If performance metric sampling is enabled dynamically with `badadmin perfmon start`. You must enable it again after running `badadmin mbdrestart`. If performance metric sampling is enabled by default, `StartTime` will be reset to the point `mbatchd` is restarted.

badadmin reconfig If `SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` parameters are changed, `badadmin reconfig` is the same as `badadmin mbdrestart`.

Performance metric logging in `lsb.streams`

By default, collected metrics must be written to `lsb.streams`. However, performance metric can still be turned on even if `ENABLE_EVENT_STREAM=N` is defined. In this case, no metric data will be logged.

- ◆ If `EVENT_STREAM_FILE` is defined and is valid, collected metrics should be written to `EVENT_STREAM_FILE`.
- ◆ If `ENABLE_EVENT_STREAM=N` is defined, metrics data will not be logged.

Job arrays

Only one submission request is counted. Element jobs are counted for jobs submitted, jobs dispatched, and jobs completed.

Job rerun

Job rerun occurs when execution hosts become unavailable while a job is running, and the job will be put to its original queue first and later will be dispatched when a suitable host is available. So in this case, only one submission request, one job submitted, and n jobs dispatched, n jobs completed are counted (n represents the number of times the job reruns before it finishes successfully).

Job requeue

Requeued jobs may be dispatched, run, and exit due to some special errors again and again. The job data always exists in the memory, so LSF only counts one job submission request and one job submitted, and counts more than one job dispatched.

For jobs completed, if a job is requeued with `brequeue`, LSF counts two jobs completed, since requeuing a job first kills the job and later puts the job into pending list. If the job is automatically requeued, LSF counts one job completed when the job finishes successfully.

Job replay

When job replay is finished, submitted jobs are not counted in job submission and job submitted, but are counted in job dispatched and job finished.

Reporting

Reporting is a feature of Platform LSF. It allows you to look at the overall statistics of your entire cluster. You can analyze the history of hosts, resources, and workload in your cluster to get an overall picture of your cluster's performance.

IMPORTANT: Platform EGO must be enabled in the LSF cluster (`LSF_ENABLE_EGO=Y` in `lsf.conf`) to use the LSF reporting feature.

Contents

- ◆ [Introduction to Reporting](#) on page 597
- ◆ [Getting Started with Standard Reports](#) on page 598
- ◆ [Custom Reports](#) on page 600
- ◆ [System Description](#) on page 604
- ◆ [Reports Administration](#) on page 606
- ◆ [Test the Reporting Feature](#) on page 617
- ◆ [Disable the Reporting Feature](#) on page 618
- ◆ [Move to a Production Database](#) on page 619

Introduction to Reporting

An efficient cluster maximizes the usage of resources while minimizing the average wait time of workload. To ensure that your cluster is running efficiently at all times, you need to analyze the activity within your cluster to see if there are any areas for improvement.

The reporting feature uses the data loader controller service, the job data transformer service, and the data purger service to collect data from the cluster, and to maintain this data in a relational database system. The reporting feature collects the cluster data from a relational database system and displays it in reports either graphically or in tables. You can use these reports to analyze and improve the performance of your cluster, and to troubleshoot configuration problems.

You can access the reporting feature from the Console (Platform Management Console).

Standard and custom reports

Platform has provided a set of standard reports to allow you to immediately analyze your cluster without having to create any new reports. These standard reports provide the most common and useful data to analyze your cluster.

You may also create custom reports to perform advanced queries and reports beyond the data produced in the standard reports.

The database

The reporting feature optionally includes the Apache Derby database, a JDBC-based relational database system. The Derby database is a small-footprint, open source database, and is only appropriate for demo clusters. If you want to use the reporting feature to produce regular reports for a production cluster, you must use a supported commercial database.

The reporting feature supports Oracle 9i, Oracle 10g, and MySQL 5.x databases.

IMPORTANT: The Apache Derby database is not supported for any production clusters.

Getting Started with Standard Reports

For your convenience, Platform has provided several standard reports for you to use. These reports allow you to keep track of some useful statistics in your cluster.

Standard reports overview

Standard reports are based on raw data stored in the relational database, and do not perform any data aggregation or calculations.

The following is a list of the standard reports that are included with the reporting feature. For further details on a report, open its full description as described in [View the full description of a report](#) on page 599.

Name	Description	Category
Cluster Availability - EGO	EGO host availability in a cluster.	EGO
Host Resource Usage	Resource usage trends for selected hosts.	EGO
Resource Allocation vs Resource Plan	Actual resource allocation compared to resource plan and unsatisfied resource demand for the selected consumer.	EGO
Cluster Job Hourly Throughput	Number of submitted, exited, and done jobs in a cluster.	LSF
Active Job States Statistics by Queue	Number of active jobs in each active job state in a selected queue.	LSF
Cluster Availability - LSF	LSF host availability in an LSF cluster.	LSF
Cluster Job Slot Utilization	Job slot utilization levels in your cluster.	LSF
Job Slot Usage by Application Tag	Job slots used by applications as indicated by the application tag.	LSF
Service Level Agreement (SLA)	Job statistics by job state over time, compared with SLA goals.	LSF

Name	Description	Category
Hourly Desktop Job Throughput	Number of downloaded and completed jobs for each MED host or the entire cluster. You can only produce this report if you use LSF Desktop.	LSF Desktop
Desktop Utilization	Desktop utilization at each MED host or the entire cluster. You can only produce this report if you use LSF Desktop.	LSF Desktop
License Usage	The license usage under License Scheduler. You can only produce this report if you use LSF License Scheduler.	LSF License Scheduler
Jobs Forwarded to Other Clusters	The number of jobs forwarded from your cluster to other clusters. You can only produce this report if you use LSF MultiCluster.	LSF MultiCluster
Jobs Received from Other Clusters	The number of jobs forwarded to your cluster from other clusters. You can only produce this report if you use LSF MultiCluster.	LSF MultiCluster

View the full description of a report

- 1 In the Console, navigate to **Reports**, then **Standard Reports**.
- 2 Click the name of your report to open it.
- 3 Click **Report properties**.

What can I do with standard reports?

Producing reports The reports stored in the system do not include actual data. Instead, the reports define what data to extract from the system, and how to display it graphically. Reports need to be produced before you can see the data. When you produce a report, you query the database and extract specific data. The amount of system overhead depends on how much data is in the report. Standard reports have configurable parameters so you can modify the report and get exactly the data that you want.

Exporting reports Data expires from the database periodically, so producing a report at a later date may return different data, or return no output at all. After you produce a report, you can keep your results by exporting the report data as comma-separated values in a CSV file. In this way you can preserve your data outside the system and integrate it with external programs, such as a spreadsheet. You can also keep your graphical results by using your browser to save the report results as an image.

Produce a standard report

- 1 In the Console, navigate to **Reports**, then **Standard Reports**.
- 2 Click the name of your report to open it.
- 3 Set the report parameters as desired. Default settings are shown, but you can modify them to suit your needs.
- 4 Click **Produce Report**.

After a short time, the resulting data is displayed graphically.

When you close the report window, you lose the contents of the report unless you export it first.

Export report data

Once you produce a report, exporting is the best way to save the data for future use. You cannot produce the same report at a later date if the data has expired from the database.

- 1 In the Console, produce and view your report.
 - 2 Click **Export Report Data**.
 - 3 In the browser dialog, specify the output path and name the exported file. In the **Save as type** field, specify "CSV".
-

Custom Reports

You can create and use custom reports if the standard reports are insufficient for your needs.

What are custom reports?

While standard reports are provided for your use by Platform, custom reports are reports you create as needed to satisfy specific reporting needs at your site.

Custom reports let you define combinations of data that are not available in the standard reports. Custom report output is always displayed in tabular format.

What can I do with custom reports?

Creating reports

The easiest way to create a custom report is to copy an existing report, then customize the SQL query string as desired. To customize the SQL query string, you may need to refer to the data schema, which describes the organization of information in the relational database. The data schema for each standard report is available in the Console by opening the report and clicking **Help**.

Even if you cannot edit SQL, saving a report as a custom report lets you re-use the report data without having to re-input the parameters in the standard report.

- If the time period is fixed, you get the same data every time you produce the report, but the report will be empty when the data expires from the database.

- If the time period is relative, you can get data for a different time period each time you produce the report.

You can also define custom reports from a blank template and input the SQL query string directly.

When you create custom reports, you can enter a category and use it to group the reports any way you want.

Deleting reports

Unlike standard reports, custom reports can be deleted. You might prefer to rename old reports (by modifying them) instead of deleting them.

Using reports

You produce custom reports and export the data in the same way as standard reports.

Data expires from the database periodically, so producing a report at a later date may return different data, or return no output at all. After you produce a report, you can keep your results by exporting the report data as comma-separated values in a CSV file. In this way you can preserve your data outside the system and integrate it with external programs, such as a spreadsheet. You can also keep your graphical results by using your browser to save the report results as an image.

If you ever want to modify parameters of a custom report, you must edit the SQL query string directly.

Create a custom report from an existing report

This method is convenient because you can extend an existing report. Examine your current standard and custom reports and select one with similar data sources or output to the new report that you want to create.

- 1 In the Console, select the report that you want to copy, with all the parameters configured as you wish to copy them.
- 2 Click **Copy to New Custom Report**.
- 3 Edit the report properties and query string as desired.
 - a In the `Report properties` section, you should give the new report a unique name. You can also modify the report summary, description, and category.
 - b In the `Report query` section, you can modify the SQL query directly. To edit the SQL query, you will need to know about the data schema of the database. For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - c To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**. This will actually produce the report, so you might want to limit your testing to a small set of data. You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
- 4 To finish, click **Create**.

To access your new custom report, navigate to **Reports** then **Custom Reports**.

Create a new custom report

Prerequisites: You must be able to construct valid query strings with Structured Query Language (SQL).

- 1 In the Console, navigate to **Reports** then **Custom Reports**.
- 2 Select **Global Actions > Create Custom Report**.

- 3 Define the report properties and query string as desired.
 - a In the `Report properties` section, specify the report name, summary, description, and category.
 - b In the `Report query` section, input your SQL query string.

For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - c To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**.

This will actually produce the report, so you might want to limit your testing to a small set of data.

You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
- 4 To finish, click **Create**.

To access your new custom report, navigate to **Reports** then **Custom Reports**.

Modify a custom report

-
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
 - 2 Click the name of your report.
 - 3 Modify the report properties and query string as desired.
 - a Edit the report properties and SQL query string.

For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - b To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**.

This will actually produce the report, so you might want to limit your testing to a small set of data.

You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
 - 4 To confirm your changes, click **Save**.

Produce a custom report

-
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
 - 2 Click the name of your report to open it.
 - 3 Click **Produce Report**.

After a short time, the resulting data is displayed in tabular format.

When you close the report window, you will lose the contents of the report unless you export it first.

Export report data

Once you produce a report, exporting is the best way to save the data for future use. You cannot produce the same report at a later date if the data has expired from the database.

-
- 1 In the Console, produce and view your report.
 - 2 Click **Export Report Data**.
 - 3 In the browser dialog, specify the output path and name the exported file. In the **Save as type** field, specify "CSV".
-

Delete a custom report

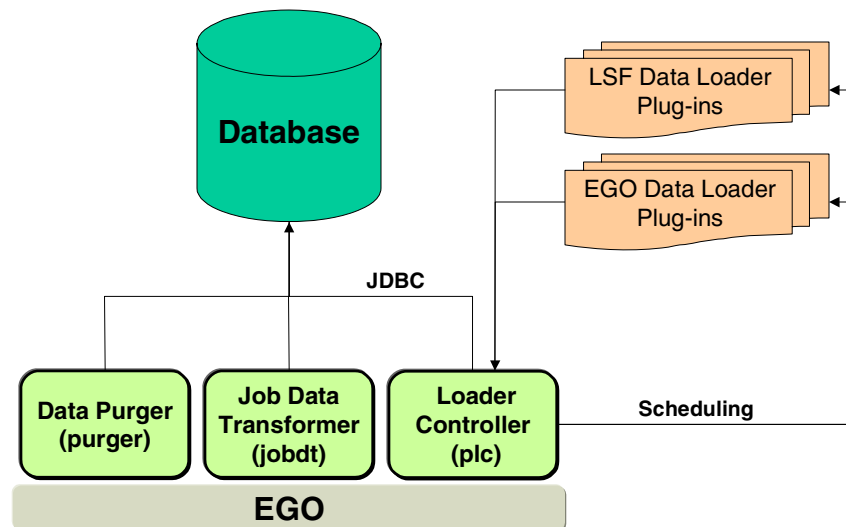
-
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
 - 2 Locate your report in the list.
 - 3 Select **Actions > Delete Report**.
-

System Description

The reporting feature is built on top of the Platform Enterprise Reporting Framework (PERF) architecture. This architecture defines the communication between your EGO cluster, relational database, and data sources via the PERF Loader Controller (PLC). The loader controller is the module that controls multiple loaders for data collection.

PERF architecture

The following diagram illustrates the PERF architecture as it relates to your cluster, reporting services, relational database, and data loader plug-ins.



LSF data loader plug-ins

The reporting feature collects LSF operation data using data loader plug-ins to load data into tables in a relational database. The data loader plug-ins connect to the database using a JDBC driver. The data loader plug-ins handle daylight savings automatically by using GMT time when collecting data.

License Scheduler

The License Scheduler data loader (`blloader`) is a polling loader that samples license usage data from License Scheduler and loads this data into the `BLD_LICUSAGE` table. This data loader is only available on Linux hosts. By default, this data loader samples data every five minutes.

Desktop job

The desktop job data loader (`desktopjobdata loader`) is a polling loader that loads job completion logs from each desktop server and loads this data into the `ACTIVE_DESKTOP_JOBDATA` table. This data loader is only available on Linux hosts. By default, this data loader loads data every day.

- Desktop client** The desktop client data loader (`desktopclientdataloader`) is a polling loader that samples client status data from the `WSCliantStatus` file and loads this data into the `ACTIVE_DESKTOP_SED_CLIENT` table. This data loader is only available on Linux hosts. By default, this data loader samples data every ten minutes.
- Desktop active event** The desktop active event data loader (`desktopenventloader`) is a polling loader that collects data on downloaded and reported jobs from the `desktop.event.log` files. For each event of type 2 (`REPORT_JOB`) and type 4 (`COMPLETE_JOB`), `desktopenventloader` loads this data into the `ACTIVE_DESKTOP_ACEVENT` table. This data loader is only available on Linux hosts. This data loader collects data when an event is logged into the `event.log` files.
- Bhosts** The bhosts data loader (`lsfbhostsloader`) is a polling loader that collects host utilization and state-related data and loads this data into the `LSF_BHOSTS` table. By default, this data loader collects data every five minutes.
- LSF events** The LSF events data loader (`lsfeventsloader`) is a polling loader that collects event data and loads this data into the `LSB_EVENTS` table. By default, this data loader collects data every five minutes.
- SLA** The SLA data loader (`lsfslaloader`) is a polling loader that samples SLA (service level agreement) performance data and loads this data into the `LSF_SLA` table. By default, this data loader samples data every five minutes.
- LSF resource properties** The LSF resource properties data loader (`lsfresproploader`) is a polling loader that samples shared resource properties data and loads this data into the `LSF_RESOURCE_PROPERTIES` table. By default, this data loader samples data every hour.
- Shared resource usage** The shared resource usage data loader (`sharedresusageloader`) is a polling loader that samples shared resource usage data and loads this data into the `SHARED_RESOURCE_USAGE` and `SHARED_RESOURCE_USAGE_HOSTLIST` tables. By default, this data loader samples data every five minutes.

EGO data loader plug-ins

The reporting feature collects EGO operation data using data loader plug-ins to load data into tables in a relational database. The data loader plug-ins connect to the database using a JDBC driver. The data loader plug-ins handle daylight savings automatically by using GMT time when collecting data.

- Consumer resource** The consumer resource data loader (`egoconsumerresloader`) is a polling loader that samples the resource allocation to each leaf consumer-resource group and loads this data into the `CONSUMER_DEMAND` and `CONSUMER_RESOURCE_ALLOCATION` tables. The consumer resource data loader also samples the host assignment to consumer data (that is, which host from which resource group is assigned to which consumer) and loads this data into the `CONSUMER_RESOURCELIST` table. By default, this data loader samples data every five minutes.

- Dynamic metric** The dynamic metric data loader (`egodynamicresloader`) is a polling loader that samples host-related dynamic metric data and loads this data into the `RESOURCE_METRICS` and `RESOURCES_RESOURCE_METRICS` tables. By default, this data loader samples data every five minutes.
- EGO allocation events** The EGO allocation events data loader (`egoeventsloader`) is a polling loader that samples allocation event data and loads this data into the `ALLOCATION_EVENT` table. By default, this data loader samples data every five minutes.
- Static attribute** The static attribute data loader (`egostaticresloader`) is a polling loader that samples host-related static attribute data and loads this data into the `RESOURCE_ATTRIBUTES` and `ATTRIBUTES_RESOURCE_METRICS` tables. By default, this data loader samples data every hour.
- System services** The reporting feature has three system services running under EGO, four system services if you are running a demo database. Each service uses one slot on a management host.
- Loader controller** The loader controller service (`plc`) controls the data loader plug-ins that collect data from the system and writes the data into the database.
- Data purger** The data purger service (`purger`) maintains the size of the database by purging old records from the data base and archiving it into `LSF_TOP/work/cluster_name/perf`. By default, the data purger purges all data that is older than 14 days, and purges data every day at 12:30am.
- Job data transformer** The job data transformer service (`jobdt`) converts raw job data in the relational database into a format usable by the reporting feature. By default, the job data transformer converts job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).
- Derby database** If you are running a demo database, the Derby database (`derbydb`) stores the cluster data. When using a supported commercial database, the Derby database service no longer runs as an EGO service.

Reports Administration

What do I need to know?

- Reports directories** The reporting feature resides in the `perf` directory, which is a subdirectory of the top-level LSF directory. This document uses `LSF_TOP` to refer to the top-level LSF installation directory and `LSF_TOP/perf` to describe the `perf` directory.
- The LSF reports configuration directory is `LSF_CONFDIR/perf/cluster_name/conf`, where `LSF_CONFDIR` refers to the LSF configuration directory (by default, `LSF_TOP/conf`).
- The LSF reports logging directory is `LSF_LOGDIR/perf`, where `LSF_LOGDIR` refers to the LSF log directory (by default, `LSF_TOP/log`).

The LSF reports working directory is `LSB_SHAREDIR/cluster_name/perf`, where `LSB_SHAREDIR` is the LSF working directory (by default, `LSF_TOP/work`).

The LSF reports data directory is `LSF_TOP\perf\data` in Windows and `LSB_SHAREDIR/cluster_name/perf/data` in Linux, where `LSB_SHAREDIR` is the LSF working directory (by default, `LSF_TOP/work`).

Reporting services

The reporting feature uses three services, which are run as EGO services.

- ◆ Job data transformer (`jobdt`)
- ◆ Loader controller (`plc`)
- ◆ Data purger (`purger`)

The Derby demo database uses the `derbydb` service, which is also run as an EGO service.

You need to stop and restart a service after editing its configuration files. If you are disabling the reporting feature, you need to disable automatic startup of these services, as described in [Disable automatic startup of the reporting services](#) on page 610.

Log files for these services are available in the `LSF_LOGDIR/perf` directory. There are seven logging levels that determine the detail of messages recorded in the log files. In decreasing level of detail, these are ALL (all messages), DEBUG, INFO, WARN, ERROR, FATAL, and OFF (no messages). By default, all service log files log messages of INFO level or higher (that is, all INFO, WARN, ERROR, and FATAL messages). You can change the logging level of the `plc` service using the loader controller client tool as described in [Dynamically change the log level of your loader controller log file](#) on page 610, or the logging level of the other services as described in [Change the log level of your log files](#) on page 611.

Job data transformer

The job data is logged in the relational database in a raw format. At regular intervals, the job data transformer converts this data to a format usable by the reporting feature. By default, the data transformer converts the job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).

To reschedule the transformation of data from the relational database to the reporting feature, you can change the data transformer schedule as described in [Change the data transformer schedule](#) on page 614.

If you edit the `jobdt.xml` configuration file under `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`, you need to restart the `jobdt` service and EGO on the master host.

Loader controller

The loader controller manages the data loader plug-ins. By default, the loader controller manages the following data loader plug-ins:

- ◆ `blloader` (License Scheduler data loader)
- ◆ `desktopjobdataloader` (Desktop job data loader)
- ◆ `desktopclientdataloader` (Desktop client data loader)
- ◆ `desktopeventloader` (Desktop active event data loader)
- ◆ `egoconsumerresloader` (consumer resource data loader)

- ◆ `egodynamicresloader` (dynamic metric data loader)
- ◆ `egoeventsloader` (EGO allocation events data loader)
- ◆ `egostaticresloader` (static attribute data loader)
- ◆ `lsfbhostsloader` (bhosts data loader)
- ◆ `lsfeventsloader` (LSF events data loader)
- ◆ `lsfslaloader` (SLA data loader)
- ◆ `lsfresproploader` (LSF resource properties data loader)
- ◆ `sharedresusageloader` (share resource usage data loader)

You can view the status of the loader controller service using the loader controller client tool as described in [View the status of the loader controller](#) on page 610.

Log files for the loader controller and data loader plug-ins are available in the `LSF_LOGDIR/perf` directory. There are seven logging levels that determine the detail of messages recorded in the log files. In decreasing level of detail, these are ALL (all messages), DEBUG, INFO, WARN, ERROR, FATAL, and OFF (no messages). By default, all service log files log messages of INFO level or higher (that is, all INFO, WARN, ERROR, and FATAL messages). You can change the logging level of the `plc` service using the loader controller client tool as described in [Dynamically change the log level of your loader controller log file](#) on page 610, or the logging level of the data loader plug-ins using the client tool as described in [Dynamically change the log level of your data loader log files](#) on page 611.

To balance data accuracy with computing power, you can change how often the data loader plug-ins collect data by changing the frequency of data collection per loader, as described in [Change the frequency of data collection](#) on page 616. To reduce the amount of unwanted data logged in the database, you can also disable individual data loader plug-ins from collecting data, as described in [Disable data collection for individual data loader plug-ins](#) on page 616.

If you edit any `plc` configuration files, you need to restart the `plc` service.

If you edit the `plc_service.xml` service configuration file under `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`, you must restart the `plc` service and EGO on the master host.

Data purger

The relational database needs to be kept to a reasonable size to maintain optimal efficiency. The data purger manages the database size by purging old data at regular intervals and archiving it into `LSF_TOP/work/cluster_name/perf`. By default, the data purger purges records older than 14 days at 12:30am every day.

To reschedule the purging of old data, you can change the purger schedule, as described in [Change the data purger schedule](#) on page 613. To reduce or increase the number of records in the database, you can change the duration of time that records are stored in the database, as described in [Change the default record expiry time](#) on page 615. If there are specific tables that are containing too much or too little data, you can also change the duration of time that records are stored in each individual table within the database, as described in [Change the record expiry time per table](#) on page 615.

If you edit any `purger` configuration files, you need to restart the `purger` service.

If you edit the `purger_service.xml` service configuration file under `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`, you must restart the purger service and EGO on the master host.

Derby database

The Derby database uses the `derbydb` service, which is run as an EGO service. You need to restart this service manually whenever you change the Derby database settings. The Derby database is only appropriate for demo clusters. To use the reporting feature to produce regular reports for a production cluster, you must move to a production database using a supported commercial database and disable automatic startup of the `derbydb` service, as described in [Move to a Production Database](#) on page 505.

Event data files

The events logger stores event data in event data files. The EGO allocation event data file is named `ego.stream` by default and has a default maximum size of 10MB. The LSF event data file is named `lsb.stream` by default and has a default maximum size of 100MB. When a data file exceeds this size, the events logger archives the file and creates a new data file.

The events logger only maintains one archive file and overwrites the old archive with the new archive. The default archive file name is `ego.stream.0` for EGO and `lsb.stream.0` for LSF. The two LSF files are located in `LSF_TOP/work/cluster_name/logdir/stream` by default, and the two EGO files are located in `LSF_TOP/work/cluster_name/ego/data` by default. The event data loaders read both the data files and the archive files.

If your system logs a large number of events, you should increase the maximum file size to see more archived event data. If your disk space is insufficient for storing the four files, you should decrease the maximum file size, or change the file path to a location with sufficient storage space. Change the disk usage of your LSF event data files as described in [Change the disk usage of LSF event data files](#) on page 612 or the file path as described in [Change the location of the LSF event data files](#) on page 612. Change the disk usage or file path of your EGO allocation event data files as described in [Change the disk usage of EGO allocation event data files](#) on page 612.

You can manage your event data files by editing the system configuration files. Edit `ego.conf` for the EGO allocation event data file configuration and `lsb.params` for the LSF event data file configuration.

Administering reports

Stop or restart reporting services

Stop or restart the `derbydb` (if you are using the Derby demo database), `jobdt.plc`, and purger services.

- 1 In the Console, navigate to **Cluster > Monitor Services**.
- 2 In your consumer tree, navigate to **Management Services > EGOManagementServices**.
- 3 Navigate to your service in the list.
- 4 If your service is running (STARTED), select **Actions > Stop** to stop it.

- 5 If you want to restart your service, select **Actions > Start**.

Disable automatic startup of the reporting services

When disabling the reporting feature, disable automatic startup of the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services. When moving from the Derby demo database to a production database, disable automatic startup of the `derbydb` service.

Disable automatic startup of these services by editing their service configuration files (`jobdt.xml`, `plc_service.xml`, `purger_service.xml`, and `derby_service.xml` for the `jobdt`, `plc`, `purger`, and `derbydb` services, respectively).

- 1 In the command console, open the EGO service directory.


```
cd LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services
```
- 2 Edit the service configuration file and change the service type from automatic to manual.

In the `<sc:StartType>` tag, change the text from `AUTOMATIC` to `MANUAL`.
- 3 Stop the service as described in [Stop or restart reporting services](#) on page 609.
- 4 In the command console, restart EGO on the master host to activate these changes.


```
egosh ego restart master_host_name
```

View the status of the loader controller

Use the loader controller client tool to view the status of the loader controller.

- 1 Launch the loader controller client tool with the `-s` option.
 - ❖ In UNIX, run `LSF_TOP/perf/version/bin/plcclient.sh -s`.
 - ❖ In Windows, run `LSF_TOP\perf\version\bin\plcclient -s`.

Dynamically change the log level of your loader controller log file

Use the loader controller client tool to dynamically change the log level of your `plc` log file if it does not cover enough detail, or covers too much, to suit your needs.

If you restart the `plc` service, the log level of your `plc` log file will be set back to the default level. To retain your new log level, change the level of your `plc` log file as described in [Change the log level of your log files](#) on page 611.

- 1 Launch the loader controller client tool with the `-l` option.
 - ❖ In UNIX, run `LSF_TOP/perf/version/bin/plcclient.sh -l log_level`.
 - ❖ In Windows, run `LSF_TOP\perf\version\bin\plcclient -l log_level`.

In decreasing level of detail, the log levels are ALL (for all messages), DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages).

Dynamically change the log level of your data loader log files

Use the loader controller client tool to dynamically change the log level of your individual data loader log files if they do not cover enough detail, or cover too much, to suit your needs.

If you restart the `plc` service, the log level of your data loader log files will be set back to the default level. To retain your new log level, change the level of your data loader log files as described in [Change the log level of your log files](#) on page 611.

- 1 If you are using the default configuration file, launch the loader controller client tool with the `-n` and `-l` options.
 - ❖ In UNIX, run `LSF_TOP/perf/version/bin/plcclient.sh -n data_loader_name -l log_level`.
 - ❖ In Windows, run `LSF_TOP\perf\version\bin\plcclient -n data_loader_name -l log_level`.

Refer to [Loader controller](#) on page 607 for a list of the data loader names.

In decreasing level of detail, the log levels are ALL (for all messages), DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages).

Change the log level of your log files

Change the log level of your log files if they do not cover enough detail, or cover too much, to suit your needs.

- 1 Edit `LSF_CONFDIR/perf/cluster_name/conf/log4j.properties`.
- 2 Navigate to the section representing the service you want to change, or to the default loader configuration if you want to change the log level of the data loader plug-ins, and look for the `*.logger.*` variable.

For example, to change the log level of the data purger log files, navigate to the following section, which is set to the default INFO level:

```
# Data purger ("purger") configuration
log4j.logger.domain.perf.purger=INFO, domain.perf.purger
```

- 3 Change the `*.logger.*` variable to the new logging level.

In decreasing level of detail, the valid values are ALL (for all messages), DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages). The services or data loader plug-ins only log messages of the same or lower level of detail as specified by the `*.logger.*` variable. Therefore, if you change the log level to ERROR, the service or data loader plug-ins will only log ERROR and FATAL messages.

For example, to change the data purger log files to the ERROR log level:

```
# Data purger ("purger") configuration
log4j.logger.domain.perf.purger=ERROR, domain.perf.purger
```

- 4 Restart the service that you changed (or the `plc` service if you changed the data loader log level) as described in [Stop or restart reporting services](#) on page 609.

Change the disk usage of LSF event data files

If your system logs a large number of events and you have sufficient disk space, increase the disk space allocated to the LSF event data files.

- 1 Edit `lsb.params` and specify or change the `MAX_EVENT_STREAM_SIZE` parameter.

```
MAX_EVENT_STREAM_SIZE = integer
```

If unspecified, this is 100 by default. Change this to the new desired file size in MB.

- 2 In the command console, reconfigure the master host to activate this change.


```
badadmin reconfig
```

Change the location of the LSF event data files

If your system logs a large number of events and you do not have enough disk space, move the LSF event data files to another location.

- 1 Edit `lsb.params` and specify or change the `EVENT_STREAM_FILE` parameter.

```
EVENT_STREAM_FILE = file_path
```

If unspecified, this is

`LSF_TOP/work/cluster_name/logdir/stream/lsb.stream` by default.

- 2 In the command console, reconfigure the master host to activate this change.


```
badadmin reconfig
```

- 3 In the command console, restart the PLC service on the master host to activate this change.

```
egosh service stop plc
```

```
egosh service start plc
```

Change the disk usage of EGO allocation event data files

If your system logs a large number of events, increase the disk space allocated to the EGO allocation event data files. If your disk space is insufficient, decrease the space allocated to the EGO allocation event data files or move these files to another location.

- 1 Edit `ego.conf`.

- a To change the size of each EGO allocation event data file, specify or change the `EGO_DATA_MAXSIZE` parameter.

```
EGO_DATA_MAXSIZE = integer
```

If unspecified, this is 10 by default. Change this to the new desired file size in MB.

- b To move the files to another location, specify or change the EGO_DATA_FILE parameter.

EGO_DATA_FILE = *file_path*

If unspecified, this is

LSF_TOP/work/cluster_name/ego/data/ego.stream by default.

- 2 In the command console, restart EGO on the master host to activate this change.

```
egosh ego restart master_host_name
```

Change the data purger schedule

To reschedule the deletion of old data, change the time in which the data purger deletes the old data.

- 1 Edit `purger_service.xml` under `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`.
- 2 Navigate to `<ego:Command>` with the `-t` parameter in the purger script.
 - ◆ In UNIX, this is `<ego:Command>/purger.sh -t ...`
 - ◆ In Windows, this is `<ego:Command>/purger.bat -t ...`

By default, the data purger is scheduled to delete old data at 12:30am every day.
- 3 Change the `-t` parameter in the data purger script to the new time (`-t new_time`).

You can change the data purger schedule to a specific daily time, or at regular time intervals, in minutes, from when the purger service first starts up.

For example, to change the schedule of the data purger:

- ◆ To delete old data at 11:15pm every day:


```
<ego:Command> ... ../purger... -t 23:15
```
 - ◆ To delete old data every 12 hours from when the purger service first starts up:


```
<ego:Command> ... ../purger... -t *[12]
```
- 4 In the command console, restart EGO on the master host to activate these changes.


```
egosh ego restart master_host_name
```
 - 5 Restart the purger service as described in [Stop or restart reporting services](#) on page 609.
-

Change the data transformer schedule

To have reschedule the transformation of data from the relational database to the reporting feature, change the time in which the data transformer converts job data.

- 1 Edit `jobdt.xml` under
`LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services.`
- 2 Navigate to `<ego:Command>` with the `-t` parameter in the purger script.
 - ◆ In UNIX, this is `<ego:Command>/jobdt.sh -t ...`
 - ◆ In Windows, this is `<ego:Command>/jobdt.bat -t ...`

By default, the data transformer converts the job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).

- 3 Change the `-t` parameter in the data transformer script to the new time (`-t new_time`).

You can change the data transformer schedule to a specific daily time, a specific hourly time, or at regular time intervals, in minutes or hours, from when the `jobdt` service first starts up.

For example, to change the schedule of the data transformer:

- ◆ To convert job data at 10:20pm every day:
`<ego:Command>/jobdt... -t 22:20`
 - ◆ To convert job data at the 25th minute of every hour:
`<ego:Command>/jobdt... -t *:25`
 - ◆ To convert job data every fifteen minutes from when the `jobdt` service first starts up:
`<ego:Command>/jobdt... -t *:[15]`
 - ◆ To convert job data every two hours from when the `jobdt` service first starts up:
`<ego:Command>/jobdt... -t *[2]`
- 4 In the command console, restart EGO on the master host to activate these changes.
`egosh ego restart master_host_name`
 - 5 Restart the `jobdt` service as described in [Stop or restart reporting services](#) on page 609.

Change the default record expiry time

To reduce or increase the number of records stored in the database, change the duration of time that a record is stored in the database before it is purged. This applies to all tables in the database unless you also specify the record expiry time in a particular table.

- 1 Edit the purger configuration files for your data loaders.
 - ❖ For EGO data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/purger/purger_ego.xml.
 - ❖ For LSF data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/purger/purger_lsf.xml.
- 2 In the `<TableList>` tag, edit the `Duration` attribute to your desired time in days, up to a maximum of 31 days.
For example, to have the records purged after 7 days:
`<TableList Duration="7">`
By default, the records are purged after 14 days.
- 3 Restart the purger service as described in [Stop or restart reporting services](#) on page 609.

Change the record expiry time per table

To reduce or increase the number of records stored in the database for a particular table, change the duration of time that a record is stored in the database per table before it is purged. The duration only applies to this particular table.

- 1 Edit the purger configuration files for your data loaders.
 - ❖ For EGO data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/purger/purger_ego.xml.
 - ❖ For LSF data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/purger/purger_lsf.xml.
- 2 Navigate to the specific `<Table>` tag with the `TableName` attribute matching the table that you want to change.
For example:
`<Table TableName="RESOURCE_METRICS"
TimestampColumn="TIME_STAMP" ... />`
- 3 Add or edit the `Duration` attribute with your desired time in days, up to a maximum of 31 days.
For example, to have the records in this table purged after 10 days:
`<Table TableName="RESOURCE_METRICS"
TimestampColumn="TIME_STAMP" Duration="10" ... />`
- 4 Restart the purger service as described in [Stop or restart reporting services](#) on page 609.

Change the frequency of data collection

To change how often the data loader plug-ins collect data, change the frequency of data collection per loader.

- 1 Edit the plc configuration files for your data loaders.
 - ❖ For EGO data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/plc/plc_ego.xml.
 - ❖ For LSF data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/plc/plc_lsf.xml.
- 2 Navigate to the specific `<DataLoader>` tag with the Name attribute matching the data loader plug-in that you want to change.
For example:
`<DataLoader Name="egodynamicresloader" Interval="300" ... />`
- 3 Add or edit the Interval attribute with your desired time in seconds.
For example, to have this plug-in collect data every 200 seconds:
`<DataLoader Name="egodynamicresloader" Interval="200" ... />`
- 4 Restart the plc service as described in [Stop or restart reporting services](#) on page 609.

Disable data collection for individual data loader plug-ins

To reduce unwanted data from being logged in the database, disable data collection for individual data loader plug-ins.

- 1 Edit the plc configuration files for your data loaders.
 - ❖ For EGO data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/plc/plc_ego.xml.
 - ❖ For LSF data loaders, edit
LSF_CONFDIR/perf/cluster_name/conf/plc/plc_lsf.xml.
- 2 Navigate to the specific `<DataLoader>` tag with the Name attribute matching the data loader plug-in that you want to disable.
For example:
`<DataLoader Name="egodynamicresloader" ... Enable="true" .../>`
- 3 Edit the Enable attribute to "false".
For example, to disable data collection for this plug-in:
`<DataLoader Name="egodynamicresloader" ... Enable="false" ... />`
- 4 Restart the plc service as described in [Stop or restart reporting services](#) on page 609.

Test the Reporting Feature

Verify that components of the reporting feature are functioning properly.

- 1 Check that the reporting services are running.
 - a In the Console, under the Service View of the Cluster Health Dashboard, check that the State of the `plc`, `purger`, and `jobdt` services are `STARTED`.
 - b If you are running the Derby demo database, check that the State of the `derbydb` service is `STARTED`.
- 2 Check that there are no error messages in the reporting logs.
 - a View the `LSF_LOGDIR/perf/plc.host_name.log` file.
 - b Verify that there are no `ERROR` messages and that, in the Data Loader Statistics section, there are data loader statistics messages for the data loader plug-ins in the last hour.

You need to find statistics messages for the following data loader plug-ins:

- ◆ `bldloader`
- ◆ `desktopjobdataloader`
- ◆ `desktopclientdataloader`
- ◆ `desktopeventloader`
- ◆ `egoconsumerresloader`
- ◆ `egodynamicresloader`
- ◆ `egoeventsloader`
- ◆ `egostaticresloader`
- ◆ `lsfbhostsloader`
- ◆ `lsfeventsloader`
- ◆ `lsfslaloader`
- ◆ `lsfresproploader`
- ◆ `sharedresusageloader`

- c View the data purger and data loader plug-in log files and verify that there are no `ERROR` messages in these files.

You need to view the following log files (`PERF_LOGDIR` is `LSF_LOGDIR/perf`):

- ◆ `PERF_LOGDIR/dataloader/bldloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/desktopjobdataloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/desktopclientdataloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/desktopeventloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/egoconsumerresloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/egodynamicresloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/egoeventsloader.host_name.log`

- ◆ `PERF_LOGDIR/dataloader/egostaticresloader.host_name.log`
- ◆ `PERF_LOGDIR/jobdt.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/lfsbhostsloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/lfsfeventsloader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/lfsflaloader.host_name.log`
- ◆ `PERF_LOGDIR/purger.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/lfsfresproploader.host_name.log`
- ◆ `PERF_LOGDIR/dataloader/sharedresusageloader.host_name.log`

- 3 Check the report output.
 - a Produce a standard report as described in [Produce a standard report](#) on page 482.
 - b Verify that the standard report produces a chart or table with data for your cluster.

Postrequisites: If you were not able to verify that these components are functioning properly, identify the cause of these problems and correct them.

Disable the Reporting Feature

Prerequisites: You must have `root` or `lsfadmin` access in the master host.

- 1 Disable the LSF events data logging.
 - a Define or edit the `ENABLE_EVENT_STREAM` parameter in the `lsb.params` file to disable event streaming.
`ENABLE_EVENT_STREAM = N`
 - b In the command console, reconfigure the master host to activate these changes.
`badmin reconfig`
- 2 Disable the EGO allocation events data logging.
 - a Define or edit the `EGO_DATA_ENABLE` parameter in the `ego.conf` file to disable data logging.
`EGO_DATA_ENABLE = N`
 - b In the command console, restart EGO on the master host to activate these changes.
`egosh ego restart master_host_name`
- 3 Stop the reporting services.
Stop the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services as described in [Stop or restart reporting services](#) on page 494.
- 4 Disable automatic startup of the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services as described in [Disable automatic startup of the reporting services](#) on page 494.

Move to a Production Database

Move the reporting feature to a production database.

Prerequisites: The commercial database is properly configured and running:

- ❑ You have a user name, password, and URL to access the database server.
- ❑ There is appropriate space in the database allocated for the reporting feature.

The Derby demo database is not supported for any production clusters. To produce regular reports for a production cluster, you must use a supported commercial database. The reporting feature supports Oracle 9i, Oracle 10g, and MySQL 5.x databases.

All data in the demo database will not be available in the production database. Some of your custom reports may not be compatible with the production database if you used non-standard SQL code.

- 1 Create a database schema for your commercial database.
 - ❖ If you are using an Oracle database, create a database schema as described in [Create an Oracle database schema](#) on page 620.
 - ❖ If you are using a MySQL database, create a database schema as described in [Create a MySQL database schema](#) on page 621.
- 2 Stop the reporting services.

Stop the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services as described in [Stop or restart reporting services](#) on page 494.
- 3 If you are using the Derby demo database, disable automatic startup of the `derbydb` service as described in [Disable automatic startup of the reporting services](#) on page 494.
- 4 If you are in UNIX, copy the Oracle JDBC driver into the PERF and GUI library directories.

You need to copy the Oracle JDBC driver to the following directories:

 - ◆ `LSF_TOP/perf/version/lib`
 - ◆ `LSF_TOP/gui/version/tomcat/common/lib`
- 5 Configure your database connection as described in [Configure the database connection](#) on page 621.
- 6 Restart the reporting services.

Restart the `jobdt`, `plc`, and `purger` services as described in [Stop or restart reporting services](#) on page 494.
- 7 Restart the Platform Management Console.

NOTE: The Platform Management Console will be unavailable during this step.

- a In the command console, stop the `WEBGUI` service.


```
egosh service stop WEBGUI
```
- b Restart the `WEBGUI` service.

```
egosh service start WEBGUI
```

The report data will now be loaded into the production database and the Console will use the data in this database.

Create an Oracle database schema

Prerequisites: The Oracle database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver (`ojdbc14.jar` or newer) for the Oracle database. This driver is available from the following URL:

http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

- 1 In the command console, open the EGO database schema directory.

```
cd LSF_TOP/perf/ego/version/DBschema/Oracle
```

- 2 Run the script to create the EGO database schema.

```
sqlplus user_name/password@connect_string @egodata.sql data_tablespace  
index_tablespace
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *connect_string* is the named SQLNet connection for this database.
- ◆ *data_tablespace* is the name of the tablespace where you intend to store the table schema.
- ◆ *index_tablespace* is the name of the tablespace where you intend to store the index.

- 3 In the command console, open the LSF database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/Oracle
```

- 4 Run the script to create the LSF database schema.

```
sqlplus user_name/password@connect_string @lsfdata.sql data_tablespace  
index_tablespace
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *connect_string* is the named SQLNet connection for this database.
- ◆ *data_tablespace* is the name of the tablespace where you intend to store the table schema.
- ◆ *index_tablespace* is the name of the tablespace where you intend to store the index.

- 5 Run the scripts to create a database schema for other installed packages.

For example, to create a database schema for LSF desktop reports,

```
sqlplus user_name/password@connect_string @lsfdesktopreport.sql data_tablespace
index_tablespace
```

Create a MySQL database schema

Prerequisites: The MySQL database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver
(mysql-connector-java-3.1.12-bin.jar or newer) for the MySQL database. This driver is available from the following URL:

<http://dev.mysql.com/downloads/>

- 1 In the command console, open the EGO database schema directory.

```
cd LSF_TOP/perf/ego/version/DBschema/MySQL
```

- 2 Run the scripts to create the EGO database schema.

```
mysql --user=user_name --password=password --database=report_database < egodata.sql
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *report_database* is the name of the database to store the report data.

- 3 In the command console, open the LSF database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/MySQL
```

- 4 Run the scripts to create the LSF database schema.

```
mysql --user=user_name --password=password --database=report_database < lsfdata.sql
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *report_database* is the name of the database to store the report data.

- 5 Run the scripts to create a database schema for other installed packages.

For example, to create a database schema for LSF desktop reports,

```
mysql --user=user_name --password=password --database=report_database <
lsfdesktopreport.sql
```

Configure the database connection

Prerequisites: You have a user name, password, and URL to access the database server.

Launch the database configuration tool to configure your database connection.

- 1 Launch the database configuration tool.
 - ❖ In UNIX (X-Windows only), run
`LSF_TOP/perf/version/bin/dbconfig.sh`.
 - ❖ In Windows, run `LSF_TOP\perf\version\bin\dbconfig`.
 - 2 In the **User ID** and **Password** fields, specify the user account name and password with which to connect to the database and to create your database tablespaces.

This user account must have been defined in your database application, and must have read and write access to the database tables. In general, this is the name you used to create your Oracle tablespaces.
 - 3 In the **JDBC driver** field, select the driver for your commercial database.
 - 4 In the **JDBC URL** field, enter the URL for your database.

This should be similar to the format given in **Example URL format**.
 - 5 In the **Maximum connections** field, specify the maximum allowed number of concurrent connections to the database server.

This is the maximum number of users who can produce reports at the same time.
-

Event Generation

Contents

- ◆ [Event Generation](#) on page 623
- ◆ [Enabling event generation](#) on page 623
- ◆ [Events list](#) on page 624
- ◆ [Arguments passed to the LSF event program](#) on page 624

Event Generation

LSF detects events occurring during the operation of LSF daemons. LSF provides a program which translates LSF events into SNMP traps. You can also write your own program that runs on the master host to interpret and respond to LSF events in other ways. For example, your program could:

- ◆ Page the system administrator
- ◆ Send email to all users
- ◆ Integrate with your existing network management software to validate and correct the problem

On Windows, use the Windows Event Viewer to view LSF events.

Enabling event generation

SNMP trap program

If you use the LSF SNMP trap program as the event handler, see the SNMP documentation for instructions on how to enable event generation.

Enable event generation for custom programs

If you use a custom program to handle the LSF events, take the following steps to enable event generation.

- 1 Write a custom program to interpret the arguments passed by LSF. See [Arguments passed to the LSF event program](#) on page 624 and [Events list](#) on page 624 for more information.
- 2 To enable event generation, define `LSF_EVENT_RECEIVER` in `lsf.conf`. You must specify an event receiver even if your program ignores it.
The event receiver maintains cluster-specific or changeable information that you do not want to hard-code into the event program. For example, the event receiver could be the path to a current log file, the email address of the cluster administrator, or the host to send SNMP traps to.
- 3 Set `LSF_EVENT_PROGRAM` in `lsf.conf` and specify the name of your custom event program. If you name your event program `genevent` (`genevent.exe` on Windows) and place it in `LSF_SERVERDIR`, you can skip this step.
- 4 Reconfigure the cluster with the commands `lsadmin reconfig` and `badmin reconfig`.

Events list

The following daemon operations cause `mbatchd` or the master LIM to call the event program to generate an event. Each LSF event is identified by a predefined number, which is passed as an argument to the event program. Events 1-9 also return the name of the host on which an event occurred.

- 1 LIM goes down (detected by the master LIM). This event may also occur if LIM temporarily stops communicating to the master LIM.
- 2 RES goes down (detected by the master LIM).
- 3 `sbatchd` goes down (detected by `mbatchd`).
- 4 An LSF server or client host becomes unlicensed (detected by the master LIM).
- 5 A host becomes the new master host (detected by the master LIM).
- 6 The master host stops being the master (detected by the master LIM).
- 7 `mbatchd` comes up and is ready to schedule jobs (detected by `mbatchd`).
- 8 `mbatchd` goes down (detected by `mbatchd`).
- 9 `mbatchd` receives a reconfiguration request and is being reconfigured (detected by `mbatchd`).
- 10 `LSB_SHAREDIR` becomes full (detected by `mbatchd`).

Arguments passed to the LSF event program

If `LSF_EVENT_RECEIVER` is defined, a function called `ls_postevent()` allows specific daemon operations to generate LSF events. This function then calls the LSF event program and passes the following arguments:

- ◆ The event receiver (LSF_EVENT_RECEIVER in `lsf.conf`)
- ◆ The cluster name
- ◆ The LSF event number (LSF events list or LSF_EVENT_XXXX macros in `lsf.h`)
- ◆ The event argument (for events that take an argument)

Example

For example, if the event receiver is the string `xxx` and LIM goes down on `HostA` in `Cluster1`, the function returns:

```
xxx Cluster1 1 HostA
```

The custom LSF event program can interpret or ignore these arguments.

Arguments passed to the LSF event program

Tuning the Cluster

Contents

- ◆ [Tuning LIM on page 628](#)
- ◆ [Improving performance of mbatchd query requests on UNIX on page 635](#)

Tuning LIM

LIM provides critical services to all LSF components. In addition to the timely collection of resource information, LIM provides host selection and job placement policies. If you are using Platform MultiCluster, LIM determines how different clusters should exchange load and resource information. You can tune LIM policies and parameters to improve performance.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host. You can also tune LIM load thresholds.

You can also change default LIM behavior and pre-select hosts to be elected master to improve performance.

In this section

- ◆ [Adjusting LIM Parameters](#) on page 628
- ◆ [Load Thresholds](#) on page 628
- ◆ [Changing Default LIM Behavior to Improve Performance](#) on page 630

Adjusting LIM Parameters

There are two main goals in adjusting LIM configuration parameters: improving response time, and reducing interference with interactive use. To improve response time, tune LSF to correctly select the best available host for each job. To reduce interference, tune LSF to avoid overloading any host.

LIM policies are advisory information for applications. Applications can either use the placement decision from LIM, or make further decisions based on information from LIM.

Most of the LSF interactive tools use LIM policies to place jobs on the network. LSF uses load and resource information from LIM and makes its own placement decisions based on other factors in addition to load information.

Files that affect LIM are `lsf.shared`, `lsf.cluster.cluster_name`, where `cluster_name` is the name of your cluster.

RUNWINDOW parameter

LIM thresholds and run windows affect the job placement advice of LIM. Job placement advice is not enforced by LIM.

The RUNWINDOW parameter defined in `lsf.cluster.cluster_name` specifies one or more time windows during which a host is considered available. If the current time is outside all the defined time windows, the host is considered locked and LIM will not advise any applications to run jobs on the host.

Load Thresholds

Load threshold parameters define the conditions beyond which a host is considered busy by LIM and are a major factor in influencing performance. No jobs will be dispatched to a busy host by LIM's policy. Each of these parameters is a load index value, so that if the host load goes beyond that value, the host becomes busy.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host.

Thresholds can be set for any load index supported internally by the LIM, and for any external load index.

If a particular load index is not specified, LIM assumes that there is no threshold for that load index. Define looser values for load thresholds if you want to aggressively run jobs on a host.

See [Load Thresholds](#) on page 523 for more details.

In this section

- ◆ [Load indices that affect LIM performance](#) on page 629
- ◆ [Comparing LIM load thresholds](#) on page 629
- ◆ [If LIM often reports a host as busy](#) on page 630
- ◆ [If interactive jobs slow down response](#) on page 630
- ◆ [Multiprocessor systems](#) on page 630

Load indices that affect LIM performance

Load index	Description
r15s	15-second CPU run queue length
r1m	1-minute CPU run queue length
r15m	15-minute CPU run queue length
pg	Paging rate in pages per second
swp	Available swap space
it	Interactive idle time
ls	Number of users logged in

For more details on load indices see [Load Indices](#) on page 237.

Comparing LIM load thresholds

To tune LIM load thresholds, compare the output of `lsload` to the thresholds reported by `lshosts -l`.

The `lsload` and `lsmom` commands display an asterisk * next to each load index that exceeds its threshold.

Example

Consider the following output from `lshosts -l` and `lsload`:

```
lshosts -l
HOST_NAME:  hostD
...
LOAD_THRESHOLDS:
   r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
   -    3.5  -    -  15  -  -  -  -   2M  1M

HOST_NAME:  hostA
...
```

Tuning LIM

LOAD_THRESHOLDS:

r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
-	3.5	-	-	15	-	-	-	-	2M	1M

lsload

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostD	ok	0.0	0.0	0.0	0%	0.0	6	0	30M	32M	10M
hostA	busy	1.9	2.1	1.9	47%	*69.6	21	0	38M	96M	60M

In this example, the hosts have the following characteristics:

- ◆ hostD is ok.
- ◆ hostA is busy—The pg (paging rate) index is 69.6, above the threshold of 15.

If LIM often reports a host as busy

If LIM often reports a host as `busy` when the CPU utilization and run queue lengths are relatively low and the system is responding quickly, the most likely cause is the paging rate threshold. Try raising the `pg` threshold.

Different operating systems assign subtly different meanings to the paging rate statistic, so the threshold needs to be set at different levels for different host types. In particular, HP-UX systems need to be configured with significantly higher `pg` values; try starting at a value of 50.

There is a point of diminishing returns. As the paging rate rises, eventually the system spends too much time waiting for pages and the CPU utilization decreases. Paging rate is the factor that most directly affects perceived interactive response. If a system is paging heavily, it feels very slow.

If interactive jobs slow down response

If you find that interactive jobs slow down system response too much while LIM still reports your host as `ok`, reduce the CPU run queue lengths (`r15s`, `r1m`, `r15m`). Likewise, increase CPU run queue lengths if hosts become busy at low loads.

Multiprocessor systems

On multiprocessor systems, CPU run queue lengths (`r15s`, `r1m`, `r15m`) are compared to the effective run queue lengths as displayed by the `lsload -E` command.

CPU run queue lengths should be configured as the load limit for a single processor. Sites with a variety of uniprocessor and multiprocessor machines can use a standard value for `r15s`, `r1m` and `r15m` in the configuration files, and the multiprocessor machines will automatically run more jobs.

Note that the normalized run queue length displayed by `lsload -N` is scaled by the number of processors. See [Load Indices](#) on page 237 and `lsfintro(1)` for the concept of effective and normalized run queue lengths.

Changing Default LIM Behavior to Improve Performance

You may want to change the default LIM behavior in the following cases:

- ◆ In very large sites. As the size of the cluster becomes large (500 hosts or more), reconfiguration of the cluster causes each LIM to re-read the configuration files. This can take quite some time.

- ◆ In sites where each host in the cluster cannot share a common configuration directory or exact replica.

In this section

- ◆ [Default LIM behavior](#) on page 631
- ◆ [Changing Default LIM Behavior to Improve Performance](#) on page 630
- ◆ [Reconfiguration and LSF_MASTER_LIST](#) on page 631
- ◆ [How LSF works with LSF_MASTER_LIST](#) on page 632
- ◆ [Considerations](#) on page 633

Default LIM behavior

By default, each LIM running in an LSF cluster must read the configuration files `lsf.shared` and `lsf.cluster.cluster_name` to obtain information about resource definitions, host types, host thresholds, etc. This includes master and slave LIMs.

This requires that each host in the cluster share a common configuration directory or an exact replica of the directory.

Change default LIM behavior

The parameter `LSF_MASTER_LIST` in `lsf.conf` allows you to identify for the LSF system which hosts can become masters. Hosts not listed in `LSF_MASTER_LIST` will be considered as slave-only hosts and will never be considered to become master.

Set LSF_MASTER_LIST (lsf.conf)

- 1 Edit `lsf.conf` and set the parameter `LSF_MASTER_LIST` to indicate hosts that are candidates to become the master host. For example:

```
LSF_MASTER_LIST="hostA hostB hostC"
```

The order in which you specify hosts in `LSF_MASTER_LIST` is the preferred order for selecting hosts to become the master LIM.

- 2 Save your changes.
- 3 Reconfigure the cluster


```
lsadmin reconfig
badmim mbdrestart.
```

Reconfiguration and LSF_MASTER_LIST

If you change LSF_MASTER_LIST

Whenever you change the parameter `LSF_MASTER_LIST`, reconfigure the cluster with `lsadmin reconfig` and `badmim mbdrestart`.

If you change `lsf.cluster.cluster_name` or `lsf.shared`

If you make changes that do not affect load report messages such as adding or removing slave-only hosts, you only need to restart the LIMs on all master candidates with the command `lsadmin limrestart` and the specific host names.

For example:

```
lsadmin limrestart hostA hostB hostC
```

If you make changes that affect load report messages such as load indices, you must restart all the LIMs in the cluster. Use the command `lsadmin reconfig`.

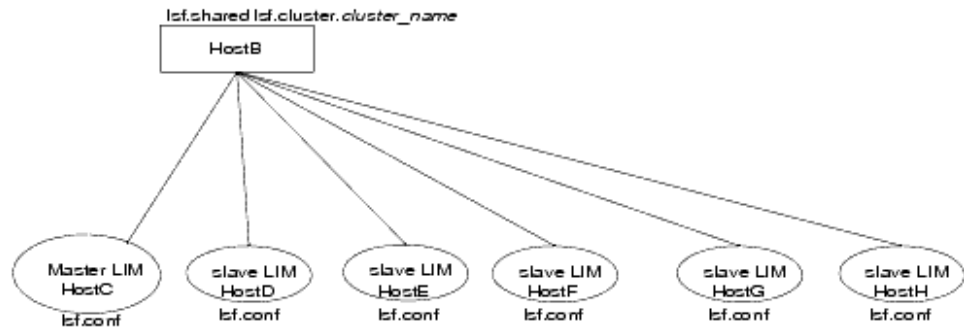
How LSF works with LSF_MASTER_LIST

LSF_MASTER_LIST undefined

In this example, `lsf.shared` and `lsf.cluster.cluster_name` are shared among all LIMs through an NFS file server. The preferred master host is the first available server host in the cluster list in `lsf.cluster.cluster_name`.

Any slave LIM can become the master LIM. Whenever you reconfigure, all LIMs read `lsf.shared` and `lsf.cluster.cluster_name` to get updated information.

For this example, slave LIMs read local `lsf.conf` files.



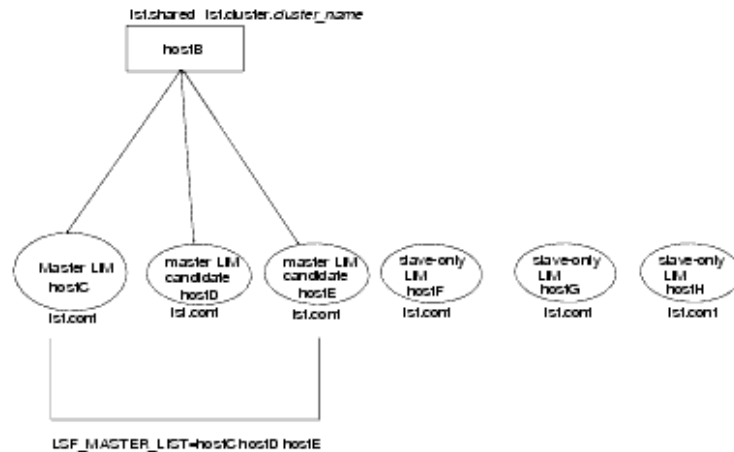
LSF_MASTER_LIST defined

The files `lsf.shared` and `lsf.cluster.cluster_name` are shared only among LIMs listed as candidates to be elected master with the parameter `LSF_MASTER_LIST`.

The preferred master host is no longer the first host in the cluster list in `lsf.cluster.cluster_name`, but the first host in the list specified by `LSF_MASTER_LIST` in `lsf.conf`.

Whenever you reconfigure, only master LIM candidates read `lsf.shared` and `lsf.cluster.cluster_name` to get updated information. The elected master LIM sends configuration information to slave LIMs.

The order in which you specify hosts in `LSF_MASTER_LIST` is the preferred order for selecting hosts to become the master LIM.



Considerations

Generally, the files `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates should be identical.

When the cluster is started up or reconfigured, LSF rereads configuration files and compares `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates.

In some cases in which identical files are not shared, files may be out of sync. This section describes situations that may arise should `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates not be identical to those of the elected master host.

LSF_MASTER_LIST not defined

When `LSF_MASTER_LIST` is not defined, LSF rejects candidate master hosts from the cluster if their `lsf.cluster.cluster_name` and `lsf.shared` files are different from the files of the elected master. Even if only comment lines are different, hosts are rejected.

A warning is logged in the log file `lim.log.master_host_name` and the cluster continues to run, but without the hosts that were rejected.

If you want the hosts that were rejected to be part of the cluster, ensure `lsf.cluster.cluster_name` and `lsf.shared` are identical for all hosts and restart all LIMs in the cluster with the command:

```
lsadmin limrestart all
```

LSF_MASTER_LIST defined

When `LSF_MASTER_LIST` is defined, LSF only rejects candidate master hosts listed in `LSF_MASTER_LIST` from the cluster if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for master candidates is different from the number of load indices in the `lsf.cluster.cluster_name` or `lsf.shared` files of the elected master.

A warning is logged in the log file `lim.log.master_host_name` and the cluster continues to run, but without the hosts that were rejected.

If you want the hosts that were rejected to be part of the cluster, ensure the number of load indices in `lsf.cluster.cluster_name` and `lsf.shared` are identical for all master candidates and restart LIMs on the master and all master candidates:

```
lsadmin limrestart hostA hostB hostC
```

LSF_MASTER_LIST defined, and master host goes down

If `LSF_MASTER_LIST` is defined and the elected master host goes down, and if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for the new elected master is different from the number of load indices in the files of the master that went down, LSF will reject all master candidates that do not have the same number of load indices in their files as the newly elected master. LSF will also reject all slave-only hosts. This could cause a situation in which only the newly elected master is considered part of the cluster.

A warning is logged in the log file `lim.log.new_master_host_name` and the cluster continues to run, but without the hosts that were rejected.

To resolve this, from the current master host, restart all LIMs:

```
lsadmin limrestart all
```

All slave-only hosts will be considered part of the cluster. Master candidates with a different number of load indices in their `lsf.cluster.cluster_name` or `lsf.shared` files will be rejected.

When the master that was down comes back up, you will have the same situation as described in [LSF_MASTER_LIST defined](#) on page 632. You will need to ensure load indices defined in `lsf.cluster.cluster_name` and `lsf.shared` for all master candidates are identical and restart LIMs on all master candidates.

Improving performance of mbatchd query requests on UNIX

You can improve `mbatchd` query performance on UNIX systems using the following methods:

- ◆ **Multithreading**—On UNIX platforms that support thread programming, you can change default `mbatchd` behavior to use multithreading and increase performance of query requests when you use the `bjobs` command. Multithreading is beneficial for busy clusters with many jobs and frequent query requests. This may indirectly increase overall `mbatchd` performance.
- ◆ **Hard CPU affinity**—You can specify the master host CPUs on which `mbatchd` child query processes can run. This improves `mbatchd` scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority `mbatchd` processes can run more efficiently.

In this section

- ◆ [How mbatchd works without multithreading](#) on page 635
- ◆ [Configure mbatchd to use multithreading](#) on page 635
- ◆ [Set a query-dedicated port for mbatchd](#) on page 637
- ◆ [Specify an expiry time for child mbatchds \(optional\)](#) on page 637
- ◆ [Specify hard CPU affinity](#) on page 637
- ◆ [Configure mbatchd to push new job information to child mbatchd](#) on page 638

How mbatchd works without multithreading

Ports

By default, `mbatchd` uses the port defined by the parameter `LSB_MBD_PORT` in `lsf.conf` or looks into the system services database for port numbers to communicate with LIM and job request commands.

It uses this port number to receive query requests from clients.

Servicing requests

For every query request received, `mbatchd` forks a child `mbatchd` to service the request. Each child `mbatchd` processes the request and then exits.

Configure mbatchd to use multithreading

When `mbatchd` has a dedicated port specified by the parameter `LSB_QUERY_PORT` in `lsf.conf`, it forks a child `mbatchd` which in turn creates threads to process query requests.

As soon as `mbatchd` has forked a child `mbatchd`, the child `mbatchd` takes over and listens on the port to process more query requests. For each query request, the child `mbatchd` creates a thread to process it.

The child `mbatchd` continues to listen to the port number specified by `LSB_QUERY_PORT` and creates threads to service requests until the job status changes, a new job is submitted, or until the time specified in `MBD_REFRESH_TIME` in `lsb.params` has passed.

Specify a time interval, in seconds, when `mbatchd` will fork a new child `mbatchd` to service query requests to keep information sent back to clients updated. A child `mbatchd` processes query requests creating threads.

MBD_REFRESH_TIME has the following syntax:

MBD_REFRESH_TIME=*seconds* [*min_refresh_time*]

where *min_refresh_time* defines the minimum time (in seconds) that the child mbatchd will stay to handle queries. The valid range is 0 - 300. The default is 5 seconds.

- ◆ If MBD_REFRESH_TIME is < *min_refresh_time*, the child mbatchd exits at MBD_REFRESH_TIME even if the job changes status or a new job is submitted before MBD_REFRESH_TIME expires.
- ◆ If MBD_REFRESH_TIME > *min_refresh_time* the child mbatchd exits at *min_refresh_time* if a job changes status or a new job is submitted before the *min_refresh_time*, or a job exits after the *min_refresh_time* when a job changes status or a new job is submitted.
- ◆ If MBD_REFRESH_TIME > *min_refresh_time* and no job changes status or a new job is submitted, the child mbatchd exits at MBD_REFRESH_TIME

The default for *min_refresh_time* is 10 seconds.

If you use the bjobs command and do not get up-to-date information, you may want to decrease the value of MBD_REFRESH_TIME or MIN_REFRESH_TIME in lsb.params to make it likely that successive job queries could get the newly-submitted job information.

NOTE: Lowering the value of MBD_REFRESH_TIME or MIN_REFRESH_TIME increases the load on mbatchd and might negatively affect performance.

- 1 Specify a query-dedicated port for the mbatchd by setting LSB_QUERY_PORT in lsf.conf.
See [Set a query-dedicated port for mbatchd](#) on page 637.
- 2 Optional: Set an interval of time to indicate when a new child mbatchd is to be forked by setting MBD_REFRESH_TIME in lsb.params. The default value of MBD_REFRESH_TIME is 5 seconds, and valid values are 0-300 seconds.
See [Specify an expiry time for child mbatchds \(optional\)](#) on page 637.
- 3 Optional: Use NEWJOB_REFRESH=Y in lsb.params to enable a child mbatchd to get up to date new job information from the parent mbatchd.
See [Configure mbatchd to push new job information to child mbatchd](#) on page 638.

Set a query-dedicated port for mbatchd

To change the default `mbatchd` behavior so that `mbatchd` forks a child `mbatchd` that can create threads, specify a port number with `LSB_QUERY_PORT` in `lsf.conf`.

TIP: This configuration only works on UNIX platforms that support thread programming.

- 1 Log on to the host as the primary LSF administrator.
- 2 Edit `lsf.conf`.
- 3 Add the `LSB_QUERY_PORT` parameter and specify a port number that will be dedicated to receiving requests from hosts.
- 4 Save the `lsf.conf` file.
- 5 Reconfigure the cluster:


```
badmin mbdrestart
```

Specify an expiry time for child mbatchds (optional)

Use `MBD_REFRESH_TIME` in `lsb.params` to define how often `mbatchd` forks a new child `mbatchd`.

- 1 Log on to the host as the primary LSF administrator.
- 2 Edit `lsb.params`.
- 3 Add the `MBD_REFRESH_TIME` parameter and specify a time interval in seconds to fork a child `mbatchd`.

The default value for this parameter is 5 seconds. Valid values are 0 to 300 seconds.
- 4 Save the `lsb.params` file.
- 5 Reconfigure the cluster as follows:


```
badmin reconfig
```

Specify hard CPU affinity

You can specify the master host CPUs on which `mbatchd` child query processes can run (hard CPU affinity). This improves `mbatchd` scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority `mbatchd` processes can run more efficiently.

When you define this parameter, LSF runs `mbatchd child` query processes *only* on the specified CPUs. The operating system can assign other processes to run on the same CPU, however, if utilization of the bound CPU is lower than utilization of the unbound CPUs.

- 1 Identify the CPUs on the master host that will run `mbatchd child` query processes.
 - ❖ Linux: To obtain a list of valid CPUs, run the command `/proc/cpuinfo`
 - ❖ Solaris: To obtain a list of valid CPUs, run the command `psrinfo`

- 2 In the file `lsb.params`, define the parameter `MBD_QUERY_CPUS`. For example, if you specify:

```
MBD_QUERY_CPUS=1 2
```

the `mbatchd child` query processes will run only on CPU numbers 1 and 2 on the master host.

You can specify CPU affinity only for master hosts that use one of the following operating systems:

- ◆ Linux 2.6 or higher
- ◆ Solaris 8 or higher

If failover to a master host candidate occurs, LSF maintains the hard CPU affinity, provided that the master host candidate has the same CPU configuration as the original master host. If the configuration differs, LSF ignores the CPU list and reverts to default behavior.

- 3 Verify that the `mbatchd child` query processes are bound to the correct CPUs on the master host.
 - a Start up a query process by running a query command such as `bjobs`.
 - b Check to see that the query process is bound to the correct CPU.
 - ◆ Linux: Run the command `taskset -p <pid>`
 - ◆ Solaris: Run the command `ps -AP`

Configure mbatchd to push new job information to child mbatchd

Prerequisites: `LSB_QUERY_PORT` must be defined. in `lsf.conf`.

If you have enabled multithreaded `mbatchd` support, the `bjobs` command may not display up-to-date information if two consecutive query commands are issued before a child `mbatchd` expires because child `mbatchd` job information is not updated. Use `NEWJOB_REFRESH=Y` in `lsb.params` to enable a child `mbatchd` to get up to date new job information from the parent `mbatchd`.

When `NEWJOB_REFRESH=Y` the parent `mbatchd` pushes new job information to a child `mbatchd`. Job queries with `bjobs` display new jobs submitted after the child `mbatchd` was created.

-
- 1 Log on to the host as the primary LSF administrator.
 - 2 Edit `lsb.params`.
 - 3 Add `NEWJOB_REFRESH=Y`.
You should set `MBD_REFRESH_TIME` in `lsb.params` to a value greater than 10 seconds.
 - 4 Save the `lsb.params` file.
 - 5 Reconfigure the cluster as follows:
`badmin reconfig`
-

Improving performance of mbatchd query requests on UNIX

Authentication and Authorization

LSF uses authentication and authorization to ensure the security of your cluster. The authentication process verifies the identity of users, hosts, and daemons, depending on the security requirements of your site. The authorization process enforces user account permissions.

Contents

- ◆ [Authentication options](#) on page 641
- ◆ [Authorization options](#) on page 643

Authentication options

During LSF installation, the authentication method is set to external authentication (`eauth`), which offers the highest level of security. To change the authentication method used by LSF, configure the parameter `LSF_AUTH` in `lsf.conf`.

Authentication options

Authentication method	Description	Configuration	Behavior
External authentication	<ul style="list-style-type: none">◆ A framework that enables you to integrate LSF with any third-party authentication product—such as Kerberos or DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site.	LSF_AUTH=eauth	<ul style="list-style-type: none">◆ LSF uses the default <code>eauth</code> executable located in <code>LSF_SERVERDIR</code>. The default executable provides an example of how the <code>eauth</code> protocol works. You should write your own <code>eauth</code> executable to meet the security requirements of your cluster. For a detailed description of the external authentication feature and how to configure it, see the <i>Platform LSF Configuration Reference</i>.
Identification daemon (<code>identd</code>)	<ul style="list-style-type: none">◆ Authentication using the <code>identd</code> daemon available in the public domain.	LSF_AUTH=ident	<ul style="list-style-type: none">◆ LSF uses the <code>identd</code> daemon available in the public domain.◆ LSF supports both RFC 931 and RFC 1413 protocols.
Privileged ports (<code>setuid</code>)	<ul style="list-style-type: none">◆ User authentication between LSF clients and servers on UNIX hosts only. An LSF command or other executable configured as <code>setuid</code> uses a reserved (privileged) port number (1-1024) to contact an LSF server. The LSF server accepts requests received on a privileged port as coming from the <code>root</code> user and then runs the LSF command or other executable using the real user account of the user who issued the command.	LSF_AUTH not defined	<ul style="list-style-type: none">◆ For UNIX hosts only, LSF clients (API functions) use reserved ports 1-1024 to communicate with LSF servers.◆ The number of user accounts that can connect concurrently to remote hosts is limited by the number of available privileged ports.◆ LSF_AUTH must be deleted or commented out and LSF commands must be installed as <code>setuid</code> programs owned by <code>root</code>.

NOTE: If you change the authentication method while LSF daemons are running, you must shut down and restart the daemons on all hosts in order to apply the changes.

When the external authentication (`eauth`) feature is enabled, you can also configure LSF to authenticate daemons by defining the parameter `LSF_AUTH_DAEMONS` in `lsf.conf`.

All authentication methods supported by LSF depend on the security of the `root` account on all hosts in the cluster.

UNIX user and host authentication

The primary LSF administrator can configure additional authentication for UNIX users and hosts by defining the parameter `LSF_USE_HOSTEQUIV` in the `lsf.conf` file. With `LSF_USE_HOSTEQUIV` defined, `mbatchd` on the master host and `RES` on the remote host call the `ruserok(3)` function to verify that the originating host is listed in the `/etc/hosts.equiv` file and that the host and user account are listed in

the `$HOME/.rhosts` file. Include the name of the local host in both files. This additional level of authentication works in conjunction with `eauth`, privileged ports (`setuid`), or `identd` authentication.

CAUTION: Using the `/etc/hosts.equiv` and `$HOME/.rhosts` files grants permission to use the `rlogin` and `rsh` commands without requiring a password.

Strict checking protocol in an untrusted environment

To improve security in an untrusted environment, the primary LSF administrator can enable the use of a strict checking communications protocol. When you define `LSF_STRICT_CHECKING` in `lsf.conf`, LSF authenticates messages passed between LSF daemons and between LSF commands and daemons. This type of authentication is *not* required in a secure environment, such as when your cluster is protected by a firewall.

IMPORTANT: You must shut down the cluster before adding or deleting the `LSF_STRICT_CHECKING` parameter.

Authentication failure

If authentication fails (the user's identity cannot be verified), LSF displays the following error message after a user issues an LSF command:

```
User permission denied
```

This error has several possible causes depending on the authentication method used.

Authentication method	Possible cause of failure
<code>eauth</code>	◆ External authentication failed
<code>identd</code>	◆ The identification daemon is not available on the local or submitting host
<code>setuid</code>	◆ The LSF applications are not installed <code>setuid</code> ◆ The NFS directory is mounted with the <code>nosuid</code> option
<code>ruserok</code>	◆ The client (local) host is not found in either the <code>/etc/hosts.equiv</code> or the <code>\$HOME/.rhosts</code> file on the master or remote host

Authorization options

Operating system authorization

By default, an LSF job or command runs on the execution host under the user account that submits the job or command, with the permissions associated with that user account. Any UNIX or Windows user account with read and execute permissions for LSF commands can use LSF to run jobs—the LSF administrator does not need to define a list of LSF users. User accounts must have the operating system permissions required to execute commands on remote hosts. When users have valid accounts on all hosts in the cluster, they can run jobs using their own account permissions on any execution host.

Authorization options

All external executables invoked by the LSF daemons, such as `esub`, `eexec`, `elim`, `eauth`, and pre- and post-execution commands, run under the `lsfadmin` user account.

Windows passwords

Windows users must register their Windows user account passwords with LSF by running the command `lspasswd`. If users change their passwords, they must use this command to update LSF. A Windows job does not run if the password is not registered in LSF. Passwords must be 31 characters or less.

For Windows password authorization in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

LSF authorization

As an LSF administrator, you have the following authorization options:

- ◆ Enable one or more types of user account mapping
- ◆ Specify the user account used to run `eauth` and `eexec` executables or pre- and post-execution commands
- ◆ Control user access to LSF resources and functionality

Enabling user account mapping

You can configure different types of user account mapping so that a job or command submitted by one user account runs on the remote host under a different user account.

Type of account mapping	Description
Between-host	Enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.
Cross-cluster	Enables cross-cluster job submission and execution for a MultiCluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.
UNIX/Windows	Enables cross-platform job submission and execution in a mixed UNIX/Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

For a detailed description of the user account mapping features and how to configure them, see the *Platform LSF Configuration Reference*.

Specifying a user account

To change the user account for ...	Define the parameter ...	In the file ...
eauth	LSF_EAUTH_USER	lsf.sudoers
eexec	LSF_EEXEC_USER	
Pre- and post execution commands	LSB_PRE_POST_EXEC_USER	

Controlling user access to LSF resources and functionality

If you want to ...	Define ...	In the file ...	Section ...
Specify the user accounts with cluster administrator privileges	ADMINISTRATORS	lsf.cluster. <i>cluster_name</i>	ClusterAdmins
Allow the <code>root</code> user to run jobs on a remote host	LSF_ROOT_REX	lsf.conf	N/A
Allow specific user accounts to use <code>@</code> for host redirection with <code>lscsh</code>	LSF_SHELL_AT_USERS	lsf.conf	N/A
Allow user accounts other than <code>root</code> to start LSF daemons	LSF_STARTUP_USERS LSF_STARTUP_PATH	lsf.sudoers	N/A

TIP: For information about how to configure the LSF daemon startup control feature, see the *Platform LSF Configuration Reference*.

Authorization failure

Symptom	Probable cause	Solution
User receives an email notification that LSF has placed a job in the USUSP state.	The job cannot run because the Windows password for the job is not registered with LSF.	The user should <ul style="list-style-type: none"> ◆ Register the Windows password with LSF using the command <code>lspasswd</code>. ◆ Use the <code>brresume</code> command to resume the suspended job.
<p>LSF displays one of the following error messages:</p> <ul style="list-style-type: none"> ◆ <code>findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname></code> ◆ <code>function: Gethostbyaddr_(<host>/<port>) failed: error</code> ◆ <code>main: Request from unknown host <host>/<port>: error</code> ◆ <code>function: Received request from non-LSF host <host>/<port></code> 	The LSF daemon does not recognize <i>host</i> as part of the cluster. These messages can occur if you add <i>host</i> to the configuration files without reconfiguring all LSF daemons.	<p>Run the following commands after adding a host to the cluster:</p> <ul style="list-style-type: none"> ◆ <code>lsadmin reconfig</code> ◆ <code>badmin mbdrestart</code> <p>If the problem still occurs, the host might have multiple addresses. Match all of the host addresses to the host name by either:</p> <ul style="list-style-type: none"> ◆ Modifying the system hosts file (<code>/etc/hosts</code>). The changes affect all software programs on your system. ◆ Creating an LSF hosts file (<code>EGO_CONFDIR/hosts</code>). Only LSF resolves the addresses to the specified host.
<ul style="list-style-type: none"> ◆ <code>doacceptconn: getpwnam(<username>@<host>/<port>) failed: error</code> ◆ <code>doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user</code> ◆ <code>authRequest: username/uid <userName>/<uid>@<host>/<port> does not exist</code> ◆ <code>authRequest: Submitter's name <clname>@<clhost> is different from name <lname> on this host</code> 	RES assumes that a user has the same UNIX user name and user ID on all LSF hosts. These messages occur if this assumption is violated.	If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same user ID and user name on all LSF hosts.
<ul style="list-style-type: none"> ◆ <code>doacceptconn: root remote execution permission denied</code> ◆ <code>authRequest: root job submission rejected</code> 	The root user tried to execute or submit a job but <code>LSF_ROOT_REX</code> is not defined in <code>lsf.conf</code> .	To allow the root user to run jobs on a remote host, define <code>LSF_ROOT_REX</code> in <code>lsf.conf</code> .

Symptom	Probable cause	Solution
<ul style="list-style-type: none"> ◆ <code>resControl: operation permission denied, uid = <uid></code> 	<p>The user with user ID <i>uid</i> is not allowed to make RES control requests. By default, only the LSF administrator can make RES control requests.</p>	<p>To allow the <code>root</code> user to make RES control requests, define <code>LSF_ROOT_REX</code> in <code>lsf.conf</code>.</p>
<ul style="list-style-type: none"> ◆ <code>do_restartReq: Failed to get hData of host <host_name>/<host_addr></code> 	<p><code>mbatchd</code> received a request from <code>sbatchd</code> on host <i>host_name</i>, but that host is not known to <code>mbatchd</code>. Either</p> <ul style="list-style-type: none"> ◆ The configuration file has been changed but <code>mbatchd</code> has not been reconfigured. ◆ <i>host_name</i> is a client host but <code>sbatchd</code> is running on that host. 	<p>To reconfigure <code>mbatchd</code>, run the command <code>badadmin reconfig</code></p> <p>To shut down <code>sbatchd</code> on <i>host_name</i>, run the command <code>badadmin hshutdown host_name</code></p>

Authorization options

Job Email and Job File Spooling

Contents

- ◆ [Mail Notification When a Job Starts](#) on page 649
- ◆ [File Spooling for Job Input, Output, and Command Files](#) on page 652

Mail Notification When a Job Starts

When a batch job completes or exits, LSF by default sends a job report by electronic mail to the submitting user account. The report includes the following information:

- ◆ Standard output (`stdout`) of the job
- ◆ Standard error (`stderr`) of the job
- ◆ LSF job information such as CPU, process and memory usage

The output from `stdout` and `stderr` are merged together in the order printed, as if the job was run interactively. The default standard input (`stdin`) file is the null device. The null device on UNIX is `/dev/null`.

`bsub` mail options

- B** Sends email to the job submitter when the job is dispatched and begins running. The default destination for email is defined by `LSB_MAILTO` in `lsf.conf`.
- u *user_name*** If you want mail sent to another user, use the `-u user_name` option to the `bsub` command. Mail associated with the job will be sent to the named user instead of to the submitting user account.
- N** If you want to separate the job report information from the job output, use the `-N` option to specify that the job report information should be sent by email.

Output and error file options (`-o output_file`, `-e error_file`, `-oo output_file`, and `-eo error_file`)

The output file created by the `-o` and `-oo` options to the `bsub` command normally contains job report information as well as the job output. This information includes the submitting user and host, the execution host, the CPU time (user plus system time) used by the job, and the exit status.

Mail Notification When a Job Starts

If you specify a `-o output_file` or `-oo output_file` option and do not specify a `-e error_file` or `-eo error_file` option, the standard output and standard error are merged and stored in `output_file`. You can also specify the standard input file if the job needs to read input from `stdin`.

NOTE: The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for `%J (job_ID)` and `%I (index_ID)`.

The output files specified by the output and error file options are created on the execution host.

See [Remote File Access](#) on page 656 for an example of copying the output file back to the submission host if the job executes on a file system that is not shared between the submission and execution hosts.

Disabling job email

If you do not want job output to be sent by mail, specify `stdout` and `stderr` as the files for the output and error options (`-o`, `-oo`, `-e`, and `-eo`). For example, the following command directs `stderr` and `stdout` to file named `/tmp/job_out`, and no email is sent.

```
bsub -o /tmp/job_out sleep 5
```

On UNIX, if you want no job output or email at all, specify `/dev/null` as the output file:

```
bsub -o /dev/null sleep 5
```

Example

The following example submits `myjob` to the night queue:

```
bsub -q night -i job_in -o job_out -e job_err myjob
```

The job reads its input from file `job_in`. Standard output is stored in file `job_out`, and standard error is stored in file `job_err`.

Size of job email

Some batch jobs can create large amounts of output. To prevent large job output files from interfering with your mail system, you can use the `LSB_MAILSIZE_LIMIT` parameter in `lsf.conf` to limit the size of the email containing the job output information.

By default, `LSB_MAILSIZE_LIMIT` is not enabled—no limit is set on size of batch job output email.

If the size of the job output email exceeds `LSB_MAILSIZE_LIMIT`, the output is saved to a file under `JOB_SPOOL_DIR`, or the default job output directory if `JOB_SPOOL_DIR` is undefined. The email informs users where the job output is located.

If the `-o` or `-oo` option of `bsub` is used, the size of the job output is not checked against `LSB_MAILSIZE_LIMIT`.

LSB_MAILSIZE environment variable

LSF sets `LSB_MAILSIZE` to the approximate size in KB of the email containing job output information, allowing a custom mail program to intercept output that is larger than desired. If you use the `LSB_MAILPROG` parameter to specify the custom mail program that can make use of the `LSB_MAILSIZE` environment variable, it is not necessary to configure `LSB_MAILSIZE_LIMIT`.

LSB_MAILSIZE is not recognized by the LSF default mail program. To prevent large job output files from interfering with your mail system, use LSB_MAILSIZE_LIMIT to explicitly set the maximum size in KB of the email containing the job information.

LSB_MAILSIZE values

The LSB_MAILSIZE environment variable can take the following values:

- ◆ A positive integer: if the output is being sent by email, LSB_MAILSIZE is set to the estimated mail size in KB.
- ◆ -1 :if the output fails or cannot be read, LSB_MAILSIZE is set to -1 and the output is sent by email using LSB_MAILPROG if specified in `lsf.conf`.
- ◆ Undefined: If you use the output or error options (`-o`, `-oo`, `-e`, or `-eo`) of `bsub`, the output is redirected to an output file. Because the output is not sent by email in this case, LSB_MAILSIZE is not used and LSB_MAILPROG is not called.

If the `-N` option is used with the output or error options of `bsub`, LSB_MAILSIZE is not set.

Directory for job output

The output and error options (`-o`, `-oo`, `-e`, and `-eo`) of the `bsub` and `bmod` commands can accept a file name or directory path. LSF creates the standard output and standard error files in this directory. If you specify only a directory path, job output and error files are created with unique names based on the job ID so that you can use a single directory for all job output, rather than having to create separate output directories for each job.

NOTE: The directory path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows.

Specifying a directory for job output

Make the final character in the path a slash (/) on UNIX, or a double backslash (\\) on Windows. If you omit the trailing slash or backslash characters, LSF treats the specification as a file name.

If the specified directory does not exist, LSF creates it on the execution host when it creates the standard error and standard output files.

By default, the output files have the following format:

Standard output

`output_directory/job_ID.out`

Standard error

`error_directory/job_ID.err`

Example

The following command creates the directory `/usr/share/lsf_out` if it does not exist, and creates the standard output file `job_ID.out` in this directory when the job completes:

```
bsub -o /usr/share/lsf_out/ myjob
```

The following command creates the directory `C:\lsf\work\lsf_err` if it does not exist, and creates the standard error file `job_ID.err` in this directory when the job completes:

```
bsub -e C:\lsf\work\lsf_err\ myjob
```

For more information

See the *Platform LSF Configuration Reference* for information about the `LSB_MAILSIZE` environment variable and the `LSB_MAILTO`, `LSB_MAILSIZE_LIMIT` parameters in `lsf.conf`, and `JOB_SPOOL_DIR` in `lsb.params`.

File Spooling for Job Input, Output, and Command Files

About job file spooling

LSF enables *spooling* of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.

You can make use of file spooling when submitting jobs with the `-is` and `-Zs` options to `bsub`. Use similar options in `bmod` to modify or cancel the spool file specification for the job. Use the file spooling options if you need to modify or remove the original job input or command files before the job completes. Removing or modifying the original input file does not affect the submitted job.

NOTE: The file path for spooling job input, output, and command files can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for `%J` (*job_ID*) and `%I` (*index_ID*).

File spooling is not supported across MultiClusters.

Specifying job input files

Use the `bsub -i input_file` and `bsub -is input_file` commands to get the standard input for the job from the file path name specified by *input_file*. The path can be an absolute path or a relative path to the current working directory. The input file can be any type of file, though it is typically a shell script text file.

LSF first checks the execution host to see if the input file exists. If the file exists on the execution host, LSF uses this file as the input file for the job.

If the file does not exist on the execution host, LSF attempts to copy the file from the submission host to the execution host. For the file copy to be successful, you must allow remote copy (`rcp`) access, or you must submit the job from a server host where RES is running. The file is copied from the submission host to a temporary file in the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, or your `$HOME/.lsbatch` directory on the execution host. LSF removes this file when the job completes.

The `-is` option of `bsub` spools the input file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, and uses the spooled file as the input file for the job.

Use the `bsub -is` command if you need to change the original input file before the job completes. Removing or modifying the original input file does not affect the submitted job.

Unless you use `-is`, you can use the special characters `%J` and `%I` in the name of the input file. `%J` is replaced by the job ID. `%I` is replaced by the index of the job in the array, if the job is a member of an array, otherwise by 0 (zero). The special characters `%J` and `%I` are not valid with the `-is` option.

Specifying a job command file (`bsub -Zs`)

Use the `bsub -Zs` command to spool a job command file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`. LSF uses the spooled file as the command file for the job.

Use the `bmod -Zs` command if you need to change the command file after the job has been submitted. Changing the original input file does not affect the submitted job. Use `bmod -Zsn` to cancel the last spooled command file and use the original spooled file.

The `bsub -Zs` option is not supported for embedded job commands because LSF is unable to determine the first command to be spooled in an embedded job command.

About the job spooling directory (`JOB_SPOOL_DIR`)

If `JOB_SPOOL_DIR` is specified in `lsb.params`:

- ◆ The job input file for `bsub -is` is spooled to `JOB_SPOOL_DIR/lsf_indir`. If the `lsf_indir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.
- ◆ The job command file for `bsub -Zs` is spooled to `JOB_SPOOL_DIR/lsf_cmddir`. If the `lsf_cmddir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

The `JOB_SPOOL_DIR` directory should be a shared directory accessible from the master host and the submission host. The directory must be readable and writable by the job submission users.

`JOB_SPOOL_DIR` can be any valid path up to a maximum length up to 4094 characters on UNIX and Linux or up to 255 characters for Windows.

`JOB_SPOOL_DIR` must be readable and writable by the job submission user, and it must be shared by the master host and the submission host. If the specified directory is not accessible or does not exist, `bsub -is` cannot write to the default directory `LSB_SHAREDIR/cluster_name/lsf_cmddir` and the job will fail.

Except for `bsub -is` and `bsub -Zs`, if `JOB_SPOOL_DIR` is not accessible or does not exist, output is spooled to the default job output directory `.lsbatch`.

For `bsub -is` and `bsub -Zs`, `JOB_SPOOL_DIR` must be readable and writable by the job submission user. If the specified directory is not accessible or does not exist, `bsub -is` and `bsub -Zs` cannot write to the default directory and the job will fail.

If `JOB_SPOOL_DIR` is not specified in `lsb.params`:

- ◆ The job input file for `bsub -is` is spooled to `LSB_SHAREDIR/cluster_name/lsf_indir`. If the `lsf_indir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

Modifying the job input file

- ◆ The job command file for `bsub -Zs` is spooled to `LSB_SHAREDIR/cluster_name/lsf_cmddir`. If the `lsf_cmddir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

If you want to use job file spooling, but do not specify `JOB_SPOOL_DIR`, the `LSB_SHAREDIR/cluster_name` directory must be readable and writable by all the job submission users. If your site does not permit this, you must manually create `lsf_indir` and `lsf_cmddir` directories under `LSB_SHAREDIR/cluster_name` that are readable and writable by all job submission users.

Modifying the job input file

Use the `-i` and `-is` options of `bmod` to specify a new job input file. The `-in` and `-isn` options cancel the last job input file modification made with either `-i` or `-is`.

Modifying the job command file

Use the `-Z` and `-Zs` options of `bmod` to modify the job command file specification. `-Z` modifies a command submitted without spooling, and `Zs` modifies a spooled command file. The `-Zsn` option of `bmod` cancels the last job command file modification made with `-Zs` and uses the original spooled command.

For more information

See the *Platform LSF Command Reference* for more information about the `bsub` and `bmod` commands.

See the Platform LSF Configuration Reference for more information about the `JOB_SPOOL_DIR` parameter in `lsb.params`, and the `LSF_TMPDIR` environment variable.

Non-Shared File Systems

Contents

- ◆ [About Directories and Files](#) on page 655
- ◆ [Using LSF with Non-Shared File Systems](#) on page 656
- ◆ [Remote File Access](#) on page 656
- ◆ [File Transfer Mechanism \(lsrctp\)](#) on page 658

About Directories and Files

LSF is designed for networks where all hosts have shared file systems, and files have the same names on all hosts.

LSF includes support for copying user data to the execution host before running a batch job, and for copying results back after the job executes.

In networks where the file systems are not shared, this can be used to give remote jobs access to local data.

Supported file systems

UNIX

On UNIX systems, LSF supports the following shared file systems:

- ◆ Network File System (NFS). NFS file systems can be mounted permanently or on demand using `automount`.
- ◆ Andrew File System (AFS)
- ◆ Distributed File System (DCE/DFS)

Windows

On Windows, directories containing LSF files can be shared among hosts from a Windows server machine.

Non-shared directories and files

LSF is usually used in networks with shared file space. When shared file space is not available, LSF can copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes. See [Remote File Access](#) on page 656 for more information.

Some networks do not share files between hosts. LSF can still be used on these networks, with reduced fault tolerance. See [Using LSF with Non-Shared File Systems](#) on page 656 for information about using LSF in a network without a shared file system.

Using LSF with Non-Shared File Systems

LSF installation

To install LSF on a cluster without shared file systems, follow the complete installation procedure on every host to install all the binaries, man pages, and configuration files.

Configuration files

After you have installed LSF on every host, you must update the configuration files on all hosts so that they contain the complete cluster configuration. Configuration files must be the same on all hosts.

Master host

You must choose one host to act as the LSF master host. LSF configuration files and working directories must be installed on this host, and the master host must be listed first in `lsf.cluster.cluster_name`.

You can use the parameter `LSF_MASTER_LIST` in `lsf.conf` to define which hosts can be considered to be elected master hosts. In some cases, this may improve performance.

For Windows password authentication in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs will run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Fault tolerance

Some fault tolerance can be introduced by choosing more than one host as a possible master host, and using NFS to mount the LSF working directory on only these hosts. All the possible master hosts must be listed first in `lsf.cluster.cluster_name`. As long as one of these hosts is available, LSF continues to operate.

Remote File Access

Using LSF with non-shared file space

LSF is usually used in networks with shared file space. When shared file space is not available, use the `bsub -f` command to have LSF copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

LSF attempts to run a job in the directory where the `bsub` command was invoked. If the execution directory is under the user's home directory, `sbatchd` looks for the path relative to the user's home directory. This handles some common configurations, such as cross-mounting user home directories with the `/net automount` option.

If the directory is not available on the execution host, the job is run in `/tmp`. Any files created by the batch job, including the standard output and error files created by the `-o` and `-e` options to `bsub`, are left on the execution host.

LSF provides support for moving user data from the submission host to the execution host before executing a batch job, and from the execution host back to the submitting host after the job completes. The file operations are specified with the `-f` option to `bsub`.

LSF uses the `lsrcp` command to transfer files. `lsrcp` contacts RES on the remote host to perform file transfer. If RES is not available, the UNIX `rcp` command is used. See [File Transfer Mechanism \(lsrcp\)](#) on page 658 for more information.

bsub -f

The `-f "[local_file operator [remote_file]]"` option to the `bsub` command copies a file between the submission host and the execution host. To specify multiple files, repeat the `-f` option.

local_file

File name on the submission host

remote_file

File name on the execution host

The files `local_file` and `remote_file` can be absolute or relative file path names. You must specify at least one file name. When the file `remote_file` is not specified, it is assumed to be the same as `local_file`. Including `local_file` without the operator results in a syntax error.

operator

Operation to perform on the file. The operator must be surrounded by white space. Valid values for `operator` are:

- > `local_file` on the submission host is copied to `remote_file` on the execution host before job execution. `remote_file` is overwritten if it exists.
- < `remote_file` on the execution host is copied to `local_file` on the submission host after the job completes. `local_file` is overwritten if it exists.
- << `remote_file` is appended to `local_file` after the job completes. `local_file` is created if it does not exist.
- ><, <> Equivalent to performing the > and then the < operation. The file `local_file` is copied to `remote_file` before the job executes, and `remote_file` is copied back, overwriting `local_file`, after the job completes. <> is the same as ><

If the submission and execution hosts have different directory structures, you must ensure that the directory where `remote_file` and `local_file` will be placed exists. LSF tries to change the directory to the same path name as the directory where the `bsub` command was run. If this directory does not exist, the job is run in your home directory on the execution host.

You should specify `remote_file` as a file name with no path when running in non-shared file systems; this places the file in the job's current working directory on the execution host. This way the job will work correctly even if the directory where the `bsub` command is run does not exist on the execution host. Be careful not to overwrite an existing file in your home directory.

File Transfer Mechanism (lsrscp)

bsub -i

If the input file specified with `bsub -i` is not found on the execution host, the file is copied from the submission host using the LSF remote file access facility and is removed from the execution host after the job finishes.

bsub -o and bsub -e

The output files specified with the `-o` and `-e` arguments to `bsub` are created on the execution host, and are not copied back to the submission host by default. You can use the remote file access facility to copy these files back to the submission host if they are not on a shared file system.

For example, the following command stores the job output in the `job_out` file and copies the file back to the submission host:

```
bsub -o job_out -f "job_out <" myjob
```

Example

To submit `myjob` to LSF, with input taken from the file `/data/data3` and the output copied back to `/data/out3`, run the command:

```
bsub -f "/data/data3 > data3" -f "/data/out3 < out3" myjob data3 out3
```

To run the job `batch_update`, which updates the `batch_data` file in place, you need to copy the file to the execution host before the job runs and copy it back after the job completes:

```
bsub -f "batch_data <>" batch_update batch_data
```

File Transfer Mechanism (lsrscp)

The LSF remote file access mechanism (`bsub -f`) uses `lsrscp` to process the file transfer. The `lsrscp` command tries to connect to RES on the submission host to handle the file transfer.

See [Remote File Access](#) on page 656 for more information about using `bsub -f`.

Limitations to lsrscp

Because LSF client hosts do not run RES, jobs that are submitted from client hosts should only specify `bsub -f` if `rscp` is allowed. You must set up the permissions for `rscp` if account mapping is used.

File transfer using `lsrscp` is not supported in the following contexts:

- ◆ If LSF account mapping is used; `lsrscp` fails when running under a different user account
- ◆ LSF client hosts do not run RES, so `lsrscp` cannot contact RES on the submission host

See [Authorization options](#) on page 643 for more information.

Workarounds

In these situations, use the following workarounds:

rcp on UNIX

If `lsrnp` cannot contact RES on the submission host, it attempts to use `rcp` to copy the file. You must set up the `/etc/hosts.equiv` or `HOME/.rhosts` file in order to use `rcp`.

See the `rcp(1)` and `rsh(1)` man pages for more information on using the `rcp` command.

Custom file transfer mechanism

You can replace `lsrnp` with your own file transfer mechanism as long as it supports the same syntax as `lsrnp`. This might be done to take advantage of a faster interconnection network, or to overcome limitations with the existing `lsrnp`. `sbatchd` looks for the `lsrnp` executable in the `LSF_BINDIR` directory as specified in the `lsf.conf` file.

File Transfer Mechanism (Isrctp)

Error and Event Logging

Contents

- ◆ [System Directories and Log Files](#) on page 661
- ◆ [Managing Error Logs](#) on page 662
- ◆ [System Event Log](#) on page 663
- ◆ [Duplicate Logging of Event Logs](#) on page 664
- ◆ [LSF Job Termination Reason Logging](#) on page 665

System Directories and Log Files

LSF uses directories for temporary work files, log files and transaction files and spooling.

LSF keeps track of all jobs in the system by maintaining a transaction log in the work subtree. The LSF log files are found in the directory

`LSB_SHAREDIR/cluster_name/logdir`.

The following files maintain the state of the LSF system:

`lsb.events`

LSF uses the `lsb.events` file to keep track of the state of all jobs. Each job is a transaction from job submission to job completion. LSF system keeps track of everything associated with the job in the `lsb.events` file.

`lsb.events.n`

The events file is automatically trimmed and old job events are stored in `lsb.event.n` files. When `mbatchd` starts, it refers only to the `lsb.events` file, not the `lsb.events.n` files. The `bhist` command can refer to these files.

Job script files in the info directory

When a user issues a `bsub` command from a shell prompt, LSF collects all of the commands issued on the `bsub` line and spools the data to `mbatchd`, which saves the `bsub` command script in the info directory (or in one of its subdirectories if

Managing Error Logs

`MAX_INFO_DIRS` is defined in `lsb.params` for use at dispatch time or if the job is rerun. The info directory is managed by LSF and should not be modified by anyone.

Log directory permissions and ownership

Ensure that the permissions on the `LSF_LOGDIR` directory to be writable by `root`. The LSF administrator must own `LSF_LOGDIR`.

Support for UNICOS accounting

In Cray UNICOS environments, LSF writes to the Network Queuing System (NQS) accounting data file, `nqacct`, on the execution host. This lets you track LSF jobs and other jobs together, through NQS.

Support for IRIX Comprehensive System Accounting (CSA)

The IRIX 6.5.9 Comprehensive System Accounting facility (CSA) writes an accounting record for each process in the `pacct` file, which is usually located in the `/var/adm/acct/day` directory. IRIX system administrators then use the `csabuild` command to organize and present the records on a job by job basis.

The `LSF_ENABLE_CSA` parameter in `lsf.conf` enables LSF to write job events to the `pacct` file for processing through CSA. For LSF job accounting, records are written to `pacct` at the start and end of each LSF job.

See the *Platform LSF Configuration Reference* for more information about the `LSF_ENABLE_CSA` parameter.

See the IRIX 6.5.9 resource administration documentation for information about CSA.

Managing Error Logs

Error logs maintain important information about LSF operations. When you see any abnormal behavior in LSF, you should first check the appropriate error logs to find out the cause of the problem.

LSF log files grow over time. These files should occasionally be cleared, either by hand or using automatic scripts.

Daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The error log file names for the LSF system daemons are:

- ◆ `res.log.host_name`
- ◆ `sbatchd.log.host_name`
- ◆ `mbatchd.log.host_name`
- ◆ `mbschd.log.host_name`

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons (except LIM) is controlled by the parameter `LSF_LOG_MASK` in `lsf.conf`. Possible values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

IMPORTANT: `LSF_LOG_MASK` in `lsf.conf` no longer specifies LIM logging level in LSF Version 7. For LIM, you must use `EGO_LOG_MASK` in `ego.conf` to control message logging for LIM. The default value for `EGO_LOG_MASK` is `LOG_WARNING`.

Error logging

If the optional `LSF_LOGDIR` parameter is defined in `lsf.conf`, error messages from LSF servers are logged to files in this directory.

If `LSF_LOGDIR` is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.

If `LSF_LOGDIR` is not defined, errors are logged to the system error logs (`syslog`) using the `LOG_DAEMON` facility. `syslog` messages are highly configurable, and the default configuration varies widely from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`.

If the error log is managed by `syslog`, it is probably already being automatically cleared.

If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to `syslog`. If you cannot find any error messages in the log files, they are likely in the `syslog`.

System Event Log

The LSF daemons keep an event log in the `lsb.events` file. The `mbatchd` daemon uses this information to recover from server failures, host reboots, and `mbatchd` restarts. The `lsb.events` file is also used by the `bhist` command to display detailed information about the execution history of batch jobs, and by the `badmin` command to display the operational history of hosts, queues, and daemons.

By default, `mbatchd` automatically backs up and rewrites the `lsb.events` file after every 1000 batch job completions. This value is controlled by the `MAX_JOB_NUM` parameter in the `lsb.params` file. The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`. LSF never deletes these files. If disk storage is a concern, the LSF administrator should arrange to archive or remove old `lsb.events.n` files periodically.

CAUTION: Do not remove or modify the current `lsb.events` file. Removing or modifying the `lsb.events` file could cause batch jobs to be lost.

Duplicate Logging of Event Logs

To recover from server failures, host reboots, or `mbatchd` restarts, LSF uses information stored in `lsb.events`. To improve the reliability of LSF, you can configure LSF to maintain copies of these logs, to use as a backup.

If the host that contains the primary copy of the logs fails, LSF will continue to operate using the duplicate logs. When the host recovers, LSF uses the duplicate logs to update the primary copies.

How duplicate logging works

By default, the event log is located in `LSB_SHAREDIR`. Typically, `LSB_SHAREDIR` resides on a reliable file server that also contains other critical applications necessary for running jobs, so if that host becomes unavailable, the subsequent failure of LSF is a secondary issue. `LSB_SHAREDIR` must be accessible from all potential LSF master hosts.

When you configure duplicate logging, the duplicates are kept on the file server, and the primary event logs are stored on the first master host. In other words, `LSB_LOCALDIR` is used to store the primary copy of the batch state information, and the contents of `LSB_LOCALDIR` are copied to a replica in `LSB_SHAREDIR`, which resides on a central file server. This has the following effects:

- ◆ Creates backup copies of `lsb.events`
- ◆ Reduces the load on the central file server
- ◆ Increases the load on the LSF master host

Failure of file server If the file server containing `LSB_SHAREDIR` goes down, LSF continues to process jobs. Client commands such as `bhist`, which directly read `LSB_SHAREDIR` will not work.

When the file server recovers, the current log files are replicated to `LSB_SHAREDIR`.

Failure of first master host If the first master host fails, the primary copies of the files (in `LSB_LOCALDIR`) become unavailable. Then, a new master host is selected. The new master host uses the duplicate files (in `LSB_SHAREDIR`) to restore its state and to log future events. There is no duplication by the second or any subsequent LSF master hosts.

When the first master host becomes available after a failure, it will update the primary copies of the files (in `LSB_LOCALDIR`) from the duplicates (in) and continue operations as before.

If the first master host does not recover, LSF will continue to use the files in `LSB_SHAREDIR`, but there is no more duplication of the log files.

Simultaneous failure of both hosts If the master host containing `LSB_LOCALDIR` and the file server containing `LSB_SHAREDIR` both fail simultaneously, LSF will be unavailable.

Network partitioning We assume that Network partitioning does not cause a cluster to split into two independent clusters, each simultaneously running `mbatchd`.

This may happen given certain network topologies and failure modes. For example, connectivity is lost between the first master, M1, and both the file server and the secondary master, M2. Both M1 and M2 will run `mbatchd` service with M1 logging events to `LSB_LOCALDIR` and M2 logging to `LSB_SHAREDIR`. When connectivity is restored, the changes made by M2 to `LSB_SHAREDIR` will be lost when M1 updates `LSB_SHAREDIR` from its copy in `LSB_LOCALDIR`.

The archived event files are only available on `LSB_LOCALDIR`, so in the case of network partitioning, commands such as `bhist` cannot access these files. As a precaution, you should periodically copy the archived files from `LSB_LOCALDIR` to `LSB_SHAREDIR`.

Setting an event update interval

If NFS traffic is too high and you want to reduce network traffic, use `EVENT_UPDATE_INTERVAL` in `lsb.params` to specify how often to back up the data and synchronize the `LSB_SHAREDIR` and `LSB_LOCALDIR` directories.

The directories are always synchronized when data is logged to the files, or when `mbatchd` is started on the first LSF master host.

Automatic archiving and duplicate logging

Event logs

Archived event logs, `lsb.events.n`, are not replicated to `LSB_SHAREDIR`. If LSF starts a new event log while the file server containing `LSB_SHAREDIR` is down, you might notice a gap in the historical data in `LSB_SHAREDIR`.

Configure duplicate logging

To enable duplicate logging, set `LSB_LOCALDIR` in `lsf.conf` to a directory on the first master host (the first host configured in `lsf.cluster.cluster_name`) that will be used to store the primary copies of `lsb.events`. This directory should only exist on the first master host.

- 1 Edit `lsf.conf` and set `LSB_LOCALDIR` to a local directory that exists only on the first master host.
- 2 Use the commands `lsadmin reconfig` and `badmin mbdrestart` to make the changes take effect.

LSF Job Termination Reason Logging

When a job finishes, LSF reports the last job termination action it took against the job and logs it into `lsb.acct`.

If a running job exits because of node failure, LSF sets the correct exit information in `lsb.acct`, `lsb.events`, and the job output file.

View logged job exit information (`bacct -l`)

- 1 Use `bacct -l` to view job exit information logged to `lsb.acct`:

```
bacct -l 7265
```

```
Accounting information about jobs that are:
```

LSF Job Termination Reason Logging

- submitted by all users.
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.

```
-----  
Job <7265>, User <lfsadmin>, Project <default>, Status <EXIT>, Queue <normal>,  
      Command <srun sleep 100000>  
Thu Sep 16 15:22:09: Submitted from host <hostA>, CWD <${HOME}>;  
Thu Sep 16 15:22:20: Dispatched to 4 Hosts/Processors <4*hostA>;  
Thu Sep 16 15:22:20: slurm_id=21793;ncpus=4;slurm_alloc=n[13-14];  
Thu Sep 16 15:23:21: Completed <exit>; TERM_RUNLIMIT: job killed after reaching  
      LSF run time limit.
```

Accounting information about this job:

Share group charged </lfsadmin>

CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.04	11	72	exit	0.0006	OK	OK

SUMMARY: (time unit: second)

Total number of done jobs:	0	Total number of exited jobs:	1
Total CPU time consumed:	0.0	Average CPU time consumed:	0.0
Maximum CPU time of a job:	0.0	Minimum CPU time of a job:	0.0
Total wait time in queues:	11.0		
Average wait time in queue:	11.0		
Maximum wait time in queue:	11.0	Minimum wait time in queue:	11.0
Average turnaround time:	72 (seconds/job)		
Maximum turnaround time:	72	Minimum turnaround time:	72
Average hog factor of a job:	0.00 (cpu time / turnaround time)		
Maximum hog factor of a job:	0.00	Minimum hog factor of a job:	0.00

Termination reasons displayed by bacct

When LSF detects that a job is terminated, `bacct -l` displays one of the following termination reasons:

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_ADMIN	Job killed by root or LSF administrator	15
TERM_BUCKET_KILL	Job killed with bkill -b	23
TERM_CHKPNT	Job killed after checkpointing	13
TERM_CPULIMIT	Job killed after reaching LSF CPU usage limit	12
TERM_CWD_NOTEXIST	Current working directory is not accessible or does not exist on the execution host	25
TERM_DEADLINE	Job killed after deadline expires	6
TERM_EXTERNAL_SIGNAL	Job killed by a signal external to LSF	17
TERM_FORCE_ADMIN	Job killed by root or LSF administrator without time for cleanup	9
TERM_FORCE_OWNER	Job killed by owner without time for cleanup	8
TERM_LOAD	Job killed after load exceeds threshold	3
TERM_MEMLIMIT	Job killed after reaching LSF memory usage limit	16
TERM_OTHER	Member of a chunk job in WAIT state killed and requeued after being switched to another queue.	4
TERM_OWNER	Job killed by owner	14
TERM_PREEMPT	Job killed after preemption	1
TERM_PROCESSLIMIT	Job killed after reaching LSF process limit	7
TERM_REQUEUE_ADMIN	Job killed and requeued by root or LSF administrator	11
TERM_REQUEUE_OWNER	Job killed and requeued by owner	10
TERM_RMS	Job exited from an RMS system error	18
TERM_RUNLIMIT	Job killed after reaching LSF run time limit	5
TERM_SLURM	Job terminated abnormally in SLURM (node failure)	22
TERM_SWAP	Job killed after reaching LSF swap usage limit	20
TERM_THREADLIMIT	Job killed after reaching LSF thread limit	21
TERM_UNKNOWN	LSF cannot determine a termination reason—0 is logged but TERM_UNKNOWN is not displayed	0
TERM_WINDOW	Job killed after queue run window closed	2
TERM_ZOMBIE	Job exited while LSF is not available	19

TIP: The integer values logged to the JOB_FINISH event in `lsb.acct` and termination reason keywords are mapped in `lsbatch.h`.

Restrictions

- ◆ If a queue-level JOB_CONTROL is configured, LSF cannot determine the result of the action. The termination reason only reflects what the termination reason could be in LSF.
- ◆ LSF cannot be guaranteed to catch any external signals sent directly to the job.
- ◆ In MultiCluster, a `brequeue` request sent from the submission cluster is translated to TERM_OWNER or TERM_ADMIN in the remote execution cluster. The termination reason in the email notification sent from the execution cluster as well as that in the `lsb.acct` is set to TERM_OWNER or TERM_ADMIN.

Example output of `bacct` and `bhist`

Example termination cause	Termination reason in <code>bacct -l</code>	Example <code>bhist</code> output
<code>bkill -s KILL</code> <code>bkill job_ID</code>	Completed <exit>; TERM_OWNER or TERM_ADMIN	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
<code>bkill -r</code>	Completed <exit>; TERM_FORCE_ADMIN or TERM_FORCE_OWNER when <code>sbatchd</code> is not reachable. Otherwise, TERM_USER or TERM_ADMIN	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
TERMINATE_WHEN	Completed <exit>; TERM_LOAD/ TERM_WINDOWS/ TERM_PREEMPT	Thu Mar 13 17:33:16: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:33:18: Exited by signal 2. The CPU time used is 0.1 seconds;
Memory limit reached	Completed <exit>; TERM_MEMLIMIT	Thu Mar 13 19:31:13: Exited by signal 2. The CPU time used is 0.1 seconds;
Run limit reached	Completed <exit>; TERM_RUNLIMIT	Thu Mar 13 20:18:32: Exited by signal 2. The CPU time used is 0.1 seconds.
CPU limit	Completed <exit>; TERM_CPULIMIT	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Swap limit	Completed <exit>; TERM_SWAPLIMIT	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Regular job exits when host crashes	Rusage 0, Completed <exit>; TERM_ZOMBIE	Thu Jun 12 15:49:02: Unknown; unable to reach the execution host; Thu Jun 12 16:10:32: Running; Thu Jun 12 16:10:38: Exited with exit code 143. The CPU time used is 0.0 seconds;
<code>brequeue -r</code>	For each requeue, Completed <exit>; TERM_REQUEUE_ADMIN or TERM_REQUEUE_OWNER	Thu Mar 13 17:46:39: Signal <REQUEUE_PEND> requested by user or administrator <user2>; Thu Mar 13 17:46:56: Exited by signal 2. The CPU time used is 0.1 seconds;
<code>bchkpnt -k</code>	On the first run: Completed <exit>; TERM_CHKPNT	Wed Apr 16 16:00:48: Checkpoint succeeded (actpid 931249); Wed Apr 16 16:01:03: Exited with exit code 137. The CPU time used is 0.0 seconds;
<code>Kill -9 <RES></code> and job	Completed <exit>; TERM_EXTERNAL_SIGNAL	Thu Mar 13 17:30:43: Exited by signal 15. The CPU time used is 0.1 seconds;
Others	Completed <exit>;	Thu Mar 13 17:30:43: Exited with 3; The CPU time used is 0.1 seconds;

Understanding LSF job exit codes

LSF monitors a job while running and returns the exit code returned from the job itself. LSF collects this exit code via `wait3()` system call on UNIX platforms. The exit code is a result of the system exit values. Use `bhist` to see the exit code for your job.

For instance, if your application had an explicit exit 129, you would see Exit code 129 in your output. When you send a signal that terminates the job, `bhist` reports either the signal or the value of `signal+128`. If the return status is greater than 128 and the job was terminated with a signal, then `return_status-128=signal`.

Example

For return status 133, the job was terminated with signal 5 (SIGTRAP on most systems, $133-128=5$). For return status 139, the job was terminated with signal 11 (SIGSEGV on most systems, $139-128=11$), which would mean the application had a core dump. Note that these terminate signals are operating-system dependent, so signal 5 may not be SIGTRAP on and 11 may not be SIGSEGV on all UNIX systems.

For more detailed information, see Chapter 50, “[Understanding Platform LSF Job Exit Information](#)”.

LSF Job Termination Reason Logging

Troubleshooting and Error Messages

Contents

- ◆ [Shared File Access](#) on page 672
- ◆ [Common LSF Problems](#) on page 673
- ◆ [Error Messages](#) on page 680
- ◆ [Setting Daemon Message Log to Debug Level](#) on page 687
- ◆ [Setting Daemon Timing Levels](#) on page 690

Shared File Access

A frequent problem with LSF is non-accessible files due to a non-uniform file space. If a task is run on a remote host where a file it requires cannot be accessed using the same name, an error results. Almost all interactive LSF commands fail if the user's current working directory cannot be found on the remote host.

Shared files on UNIX

If you are running NFS, rearranging the NFS mount table may solve the problem. If your system is running the `automount` server, LSF tries to map the filenames, and in most cases it succeeds. If shared mounts are used, the mapping may break for those files. In such cases, specific measures need to be taken to get around it.

The automount maps must be managed through NIS. When LSF tries to map filenames, it assumes that automounted file systems are mounted under the `/tmp_mnt` directory.

Shared files on Windows

-
- 1 To share files among Windows machines, set up a share on the server and access it from the client. You can access files on the share either by specifying a UNC path (`\\server\share\path`) or connecting the share to a local drive name and using a `drive:\path` syntax. Using UNC is recommended because drive mappings may be different across machines, while UNC allows you to unambiguously refer to a file on the network.
-

Shared files across UNIX and Windows

For file sharing across UNIX and Windows, you require a third party NFS product on Windows to export directories from Windows to UNIX.

Common LSF Problems

This section lists some other common problems with the LIM, RES, `mbatchd`, `sbatchd`, and interactive applications.

Most problems are due to incorrect installation or configuration. Check the error log files; often the log message points directly to the problem.

LIM dies quietly

- 1 Run the following command to check for errors in the LIM configuration files.

```
lsadmin ckconfig -v
```

This displays most configuration errors. If this does not report any errors, check in the LIM error log.

LIM unavailable

Sometimes the LIM is up, but executing the `lslload` command prints the following error message:

```
Communication time out.
```

If the LIM has just been started, this is normal, because the LIM needs time to get initialized by reading configuration files and contacting other LIMs. If the LIM does not become available within one or two minutes, check the LIM error log for the host you are working on.

To prevent communication timeouts when starting or restarting the local LIM, define the parameter `LSF_SERVER_HOSTS` in the `lsf.conf` file. The client will contact the LIM on one of the `LSF_SERVER_HOSTS` and execute the command, provided that at least one of the hosts defined in the list has a LIM that is up and running.

When the local LIM is running but there is no master LIM in the cluster, LSF applications display the following message:

```
Cannot locate master LIM now, try later.
```

- 1 Check the LIM error logs on the first few hosts listed in the `Host` section of the `lsf.cluster.cluster_name` file. If `LSF_MASTER_LIST` is defined in `lsf.conf`, check the LIM error logs on the hosts listed in this parameter instead.

Master LIM is down

Sometimes the master LIM is up, but executing the `lslload` or `lshosts` command prints the following error message:

```
Master LIM is down; try later
```

If the `/etc/hosts` file on the host where the master LIM is running is configured with the host name assigned to the loopback IP address (127.0.0.1), LSF client LIMs cannot contact the master LIM. When the master LIM starts up, it sets its official

host name and IP address to the loopback address. Any client requests will get the master LIM address as 127.0.0.1, and try to connect to it, and in fact will try to access itself.

-
- 1 Check the IP configuration of your master LIM in `/etc/hosts`. The following example incorrectly sets the master LIM IP address to the loopback address:

```
127.0.0.1      localhost      myhostname
```

The following example correctly sets the master LIM IP address:

```
127.0.0.1      localhost
192.168.123.123 myhostname
```

For a master LIM running on a host that uses an IPv6 address, the loopback address is

```
::1
```

The following example correctly sets the master LIM IP address using an IPv6 address:

```
::1      localhost ipv6-localhost ipv6-loopback

fe00::0      ipv6-localnet

ff00::0      ipv6-mcastprefix
ff02::1      ipv6-allnodes
ff02::2      ipv6-allrouters
ff02::3      ipv6-allhosts
```

RES does not start

-
- 1 Check the RES error log.
-

User permission denied

If remote execution fails with the following error message, the remote host could not securely determine the user ID of the user requesting remote execution.

```
User permission denied.
```

-
- 1 Check the RES error log on the remote host; this usually contains a more detailed error message.
 - 2 If you are not using an identification daemon (`LSF_AUTH` is not defined in the `lsf.conf` file), then all applications that do remote executions must be owned by root with the `setuid` bit set. This can be done as follows.

```
chmod 4755 filename
```
 - 3 If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

- 4 If you are using an identification daemon (defined in the `lsf.conf` file by `LSF_AUTH`), `inetd` must be configured to run the daemon. The identification daemon must not be run directly.
- 5 If `LSF_USE_HOSTEQUIV` is defined in the `lsf.conf` file, check if `/etc/hosts.equiv` or `HOME/.rhosts` on the destination host has the client host name in it. Inconsistent host names in a name server with `/etc/hosts` and `/etc/hosts.equiv` can also cause this problem.
- 6 On SGI hosts running a name server, you can try the following command to tell the host name lookup code to search the `/etc/hosts` file before calling the name server.


```
setenv HOSTRESORDER "local,nis,bind"
```
- 7 For Windows hosts, users must register and update their Windows passwords using the `lspasswd` command. Passwords must be 3 characters or longer, and 31 characters or less.

For Windows password authentication in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs will run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Non-uniform file name space

A command may fail with the following error message due to a non-uniform file name space.

```
chdir(...) failed: no such file or directory
```

You are trying to execute a command remotely, where either your current working directory does not exist on the remote host, or your current working directory is mapped to a different name on the remote host.

If your current working directory does not exist on a remote host, you should not execute commands remotely on that host.

On UNIX

If the directory exists, but is mapped to a different name on the remote host, you have to create symbolic links to make them consistent.

LSF can resolve most, but not all, problems using `automount`. The `automount` maps must be managed through NIS. Follow the instructions in your Release Notes for obtaining technical support if you are running `automount` and LSF is not able to locate directories on remote hosts.

Batch daemons die quietly

- 1 First, check the `sbatchd` and `mbatchd` error logs. Try running the following command to check the configuration.

```
badmin ckconfig
```

This reports most errors. You should also check if there is any email in the LSF administrator's mailbox. If the `mbatchd` is running but the `sbatchd` dies on some hosts, it may be because `mbatchd` has not been configured to use those hosts.

See [Host not used by LSF](#) on page 676.

sbatchd starts but mbatchd does not

- 1 Check whether LIM is running. You can test this by running the `lsid` command. If LIM is not running properly, follow the suggestions in this chapter to fix the LIM first. It is possible that `mbatchd` is temporarily unavailable because the master LIM is temporarily unknown, causing the following error message.

```
sbatchd: unknown service
```

- 2 Check whether services are registered properly. See [Registering Service Ports](#) on page 75 for information about registering LSF services.
-

Host not used by LSF

If you configure a list of server hosts in the `Host` section of the `lsb.hosts` file, `mbatchd` allows `sbatchd` to run only on the hosts listed. If you try to configure an unknown host in the `HostGroup` or `HostPartition` sections of the `lsb.hosts` file, or as a `HOSTS` definition for a queue in the `lsb.queues` file, `mbatchd` logs the following message.

```
mbatchd on host: LSB_CONFDIR/cluster/configdir/file(line #): Host  
hostname is not used by lsbatch;
```

```
ignored
```

If you start `sbatchd` on a host that is not known by `mbatchd`, `mbatchd` rejects the `sbatchd`. The `sbatchd` logs the following message and exits.

```
This host is not used by lsbatch system.
```

Both of these errors are most often caused by not running the following commands, in order, after adding a host to the configuration.

```
lsadmin reconfig
```

```
badmin reconfig
```

You must run both of these before starting the daemons on the new host.

UNKNOWN host type or model

Viewing UNKNOWN host type or model

- 1 Run `lshosts`. A model or type UNKNOWN indicates the host is down or the LIM on the host is down. You need to take immediate action. For example:

```
lshosts
HOST_NAME  type      model  cpuf   ncpus  maxmem  maxswp  server  RESOURCES
hostA      UNKNOWN  Ultra2  20.2   2      256M    710M    Yes    ( )
```

Fixing UNKNOWN matched host type or matched model

- 1 Start the host.
- 2 Run `lsadmin limstartup` to start LIM on the host.

For example:

```
lsadmin limstartup hostA
Starting up LIM on <hostA> .... done
```

or, if EGO is enabled in the LSF cluster, you can also run:

```
egosh ego start lim hostA
Starting up LIM on <hostA> .... done
```

You can specify more than one host name to start up LIM on multiple hosts. If you do not specify a host name, LIM is started up on the host from which the command is submitted.

On UNIX, in order to start up LIM remotely, you must be root or listed in `lsf.sudoers` (or `ego.sudoers` if EGO is enabled in the LSF cluster) and be able to run the `rsh` command across all hosts without entering a password.

- 3 Wait a few seconds, then run `lshosts` again. You should now be able to see a specific model or type for the host or DEFAULT. If you see DEFAULT, it means that automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found. LSF will work on the host, but a DEFAULT model may be inefficient because of incorrect CPU factors. A DEFAULT type may also cause binary incompatibility because a job from a DEFAULT host type can be migrated to another DEFAULT host type.

DEFAULT host type or model

Viewing DEFAULT host type or model

If you see DEFAULT in `lim -t`, it means that automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found. LSF will work on the host, but a DEFAULT model may be inefficient because of

incorrect CPU factors. A DEFAULT type may also cause binary incompatibility because a job from a DEFAULT host type can be migrated to another DEFAULT host type.

- 1 Run `lshosts`. If Model or Type are displayed as DEFAULT when you use `lshosts` and automatic host model and type detection is enabled, you can leave it as is or change it. For example:

```
lshosts
HOST_NAME      type      model      cpuf      ncpus      maxmem      maxswp      server      RESOURCES
hostA          DEFAULT  DEFAULT    1          2          256M        710M        Yes        ( )
```

If model is DEFAULT, LSF will work correctly but the host will have a CPU factor of 1, which may not make efficient use of the host model.

If type is DEFAULT, there may be binary incompatibility. For example, there are 2 hosts, one is Solaris, the other is HP. If both hosts are set to type DEFAULT, it means jobs running on the Solaris host can be migrated to the HP host and vice-versa.

Fixing DEFAULT matched host type or matched model

- 1 Run `lim -t` on the host whose type is DEFAULT:

```
lim -t
Host Type           : LINUX86
Host Architecture   : SUNWUltra2_200_sparcv9
Physical Processors : 2
Cores per Processor : 4
Threads per Core:   : 2
License Needed      : Class(B), Multi-cores
Matched Type:       DEFAULT
Matched Architecture: DEFAULT
Matched Model:      DEFAULT
CPU Factor          : 60.0
```

Note the value of Host Type and Host Architecture.

- 2 Edit `lsf.shared`.
 - a In the HostType section, enter a new host type. Use the host type name detected with `lim -t`. For example:

```
Begin HostType
TYPENAME
DEFAULT
CRAYJ
LINUX86
...
End HostType
```

- b** In the `HostModel` section, enter the new host model with architecture and CPU factor. Use the architecture detected with `lim -t`. Add the host model to the end of the host model list. The limit for host model entries is 127. Lines commented out with `#` are not counted in the 127-line limit. For example:

```
Begin HostModel
MODELNAME    CPUFACTOR    ARCHITECTURE # keyword
Ultra2       20            SUNWUltra2_200_sparcv9
End HostModel
```

- 3** Save changes to `lsf.shared`.
 - 4** Run `lsadmin reconfig` to reconfigure LIM.
 - 5** Wait a few seconds, and run `lim -t` again to check the type and model of the host.
-

Error Messages

The following error messages are logged by the LSF daemons, or displayed by the following commands.

```
lsadmin ckconfig
```

```
badmin ckconfig
```

General errors

The messages listed in this section may be generated by any LSF daemon.

```
can't open file: error
```

The daemon could not open the named file for the reason given by *error*. This error is usually caused by incorrect file permissions or missing files. All directories in the path to the configuration files must have execute (x) permission for the LSF administrator, and the actual files must have read (r) permission. Missing files could be caused by incorrect path names in the `lsf.conf` file, running LSF daemons on a host where the configuration files have not been installed, or having a symbolic link pointing to a nonexistent file or directory.

```
file(line): malloc failed
```

Memory allocation failed. Either the host does not have enough available memory or swap space, or there is an internal error in the daemon. Check the program load and available swap space on the host; if the swap space is full, you must add more swap space or run fewer (or smaller) programs on that host.

```
auth_user: getservbyname(ident/tcp) failed: error; ident must be registered in services
```

LSF_AUTH=ident is defined in the `lsf.conf` file, but the `ident/tcp` service is not defined in the services database. Add `ident/tcp` to the services database, or remove `LSF_AUTH` from the `lsf.conf` file and `setuid root` those LSF binaries that require authentication.

```
auth_user: operation(<host>/<port>) failed: error
```

LSF_AUTH=ident is defined in the `lsf.conf` file, but the LSF daemon failed to contact the `identd` daemon on host. Check that `identd` is defined in `inetd.conf` and the `identd` daemon is running on host.

```
auth_user: Authentication data format error (rbuf=<data>) from <host>/<port>
```

```
auth_user: Authentication port mismatch (...) from <host>/<port>
```

LSF_AUTH=ident is defined in the `lsf.conf` file, but there is a protocol error between LSF and the `ident` daemon on *host*. Make sure the `ident` daemon on the host is configured correctly.

```
userok: Request from bad port (<port_number>), denied
```

LSF_AUTH is not defined, and the LSF daemon received a request that originates from a non-privileged port. The request is not serviced.

Set the LSF binaries to be owned by root with the `setuid` bit set, or define `LSF_AUTH=ident` and set up an ident server on all hosts in the cluster. If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

```
userok: Forged username suspected from <host>/<port>:
<claimed_user>/<actual_user>
```

The service request claimed to come from user *claimed_user* but ident authentication returned that the user was actually *actual_user*. The request was not serviced.

```
userok: ruserok(<host>,<uid>) failed
```

`LSF_USE_HOSTEQUIV` is defined in the `lsf.conf` file, but *host* has not been set up as an equivalent host (see `/etc/host.equiv`), and user *uid* has not set up a `.rhosts` file.

```
init_AcceptSock: RES service(res) not registered, exiting
```

```
init_AcceptSock: res/tcp: unknown service, exiting
```

```
initSock: LIM service not registered.
```

```
initSock: Service lim/udp is unknown. Read LSF Guide for help
```

```
get_ports: <serv> service not registered
```

The LSF services are not registered. See [Registering Service Ports](#) on page 75 for information about configuring LSF services.

```
init_AcceptSock: Can't bind daemon socket to port <port>: error,
exiting
```

```
init_ServSock: Could not bind socket to port <port>: error
```

These error messages can occur if you try to start a second LSF daemon (for example, RES is already running, and you execute RES again). If this is the case, and you want to start the new daemon, kill the running daemon or use the `lsadmin` or `badmin` commands to shut down or restart the daemon.

Configuration errors

The messages listed in this section are caused by problems in the LSF configuration files. General errors are listed first, and then errors from specific files.

```
file(line): Section name expected after Begin; ignoring section
```

```
file(line): Invalid section name name; ignoring section
```

The keyword `begin` at the specified line is not followed by a section name, or is followed by an unrecognized section name.

```
file(line): section section: Premature EOF
```

The end of file was reached before reading the end section line for the named section.

```
file(line): keyword line format error for section section; Ignore
this section
```

The first line of the section should contain a list of keywords. This error is printed when the keyword line is incorrect or contains an unrecognized keyword.

Error Messages

```
file(line): values do not match keys for section section; Ignoring line
```

The number of fields on a line in a configuration section does not match the number of keywords. This may be caused by not putting () in a column to represent the default value.

```
file: HostModel section missing or invalid
```

```
file: Resource section missing or invalid
```

```
file: HostType section missing or invalid
```

The HostModel, Resource, or HostType section in the lsf.shared file is either missing or contains an unrecoverable error.

```
file(line): Name name reserved or previously defined. Ignoring index
```

The name assigned to an external load index must not be the same as any built-in or previously defined resource or load index.

```
file(line): Duplicate clustername name in section cluster. Ignoring current line
```

A cluster name is defined twice in the same lsf.shared file. The second definition is ignored.

```
file(line): Bad cpuFactor for host model model. Ignoring line
```

The CPU factor declared for the named host model in the lsf.shared file is not a valid number.

```
file(line): Too many host models, ignoring model name
```

You can declare a maximum of 127 host models in the lsf.shared file.

```
file(line): Resource name name too long in section resource. Should be less than 40 characters. Ignoring line
```

The maximum length of a resource name is 39 characters. Choose a shorter name for the resource.

```
file(line): Resource name name reserved or previously defined. Ignoring line.
```

You have attempted to define a resource name that is reserved by LSF or already defined in the lsf.shared file. Choose another name for the resource.

```
file(line): illegal character in resource name: name, section resource. Line ignored.
```

Resource names must begin with a letter in the set [a-zA-Z], followed by letters, digits or underscores [a-zA-Z0-9_].

LIM messages

The following messages are logged by the LIM:

```
main: LIM cannot run without licenses, exiting
```

The LSF software license key is not found or has expired. Check that `FLEXlm` is set up correctly, or contact your LSF technical support.

```
main: Received request from unlicensed host <host>/<port>
```

LIM refuses to service requests from hosts that do not have licenses. Either your LSF license has expired, or you have configured LSF on more hosts than your license key allows.

```
initLicense: Trying to get license for LIM from source
<LSF_CONFDIR/license.dat>
```

```
getLicense: Can't get software license for LIM from license file
<LSF_CONFDIR/license.dat>: feature not yet available.
```

Your LSF license is not yet valid. Check whether the system clock is correct.

```
findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname>
```

```
function: Gethostbyaddr_(<host>/<port>) failed: error
```

```
main: Request from unknown host <host>/<port>: error
```

```
function: Received request from non-LSF host <host>/<port>
```

The daemon does not recognize *host*. The request is not serviced. These messages can occur if *host* was added to the configuration files, but not all the daemons have been reconfigured to read the new information. If the problem still occurs after reconfiguring all the daemons, check whether the host is a multi-addressed host.

See [Host Naming](#) on page 77 for information about working with multi-addressed hosts.

```
rcvLoadVector: Sender (<host>/<port>) may have different config?
```

```
MasterRegister: Sender (host) may have different config?
```

LIM detected inconsistent configuration information with the sending LIM. Run the following command so that all the LIMs have the same configuration information.

```
lsadmin reconfig
```

Note any hosts that failed to be contacted.

```
rcvLoadVector: Got load from client-only host <host>/<port>. Kill
LIM on <host>/<port>
```

A LIM is running on a client host. Run the following command, or go to the client host and kill the LIM daemon.

```
lsadmin limshutdown host
```

```
saveIndx: Unknown index name <name> from ELIM
```

LIM received an external load index name that is not defined in the `lsf.shared` file. If `name` is defined in `lsf.shared`, reconfigure the LIM. Otherwise, add `name` to the `lsf.shared` file and reconfigure all the LIMs.

```
saveIndx: ELIM over-riding value of index <name>
```

This is a warning message. The ELIM sent a value for one of the built-in index names. LIM uses the value from ELIM in place of the value obtained from the kernel.

```
getusr: Protocol error numIndx not read (cc=num): error
```

```
getusr: Protocol error on index number (cc=num): error
```

Protocol error between ELIM and LIM.

RES messages

These messages are logged by the RES.

```
doacceptconn: getpwnam(<username>@<host>/<port>) failed: error
```

```
doacceptconn: User <username> has uid <uid1> on client host  
<host>/<port>, uid <uid2> on RES host; assume bad user
```

```
authRequest: username/uid <userName>/<uid>@<host>/<port> does not  
exist
```

```
authRequest: Submitter's name <cname>@<clhost> is different from  
name <lname> on this host
```

RES assumes that a user has the same userID and username on all the LSF hosts. These messages occur if this assumption is violated. If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same userID and username on all LSF hosts.

```
doacceptconn: root remote execution permission denied
```

```
authRequest: root job submission rejected
```

Root tried to execute or submit a job but LSF_ROOT_REX is not defined in the `lsf.conf` file.

```
resControl: operation permission denied, uid = <uid>
```

The user with user ID *uid* is not allowed to make RES control requests. Only the LSF manager, or root if LSF_ROOT_REX is defined in `lsf.conf`, can make RES control requests.

```
resControl: access(respath, X_OK): error
```

The RES received a reboot request, but failed to find the file `respath` to re-execute itself. Make sure `respath` contains the RES binary, and it has execution permission.

mbatchd and sbatchd messages

The following messages are logged by the `mbatchd` and `sbatchd` daemons:

```
renewJob: Job <jobId>: rename(<from>,<to>) failed: error
```

`mbatchd` failed in trying to re-submit a rerunnable job. Check that the file *from* exists and that the LSF administrator can rename the file. If *from* is in an AFS directory, check that the LSF administrator's token processing is properly setup.

See the document "Installing LSF on AFS" on the Platform Web site for more information about installing on AFS.

```
logJobInfo_: fopen(<logdir/info/jobfile>) failed: error
```

```
logJobInfo_: write <logdir/info/jobfile> <data> failed: error
```

```
logJobInfo_: seek <logdir/info/jobfile> failed: error
```

```
logJobInfo_: write <logdir/info/jobfile> xdrpos <pos> failed: error
logJobInfo_: write <logdir/info/jobfile> xdr buf len <len> failed:
error
logJobInfo_: close(<logdir/info/jobfile>) failed: error
rmLogJobInfo: Job <jobId>: can't unlink(<logdir/info/jobfile>):
error
rmLogJobInfo_: Job <jobId>: can't stat(<logdir/info/jobfile>): error
readLogJobInfo: Job <jobId> can't open(<logdir/info/jobfile>): error
start_job: Job <jobId>: readLogJobInfo failed: error
readLogJobInfo: Job <jobId>: can't read(<logdir/info/jobfile>) size
size: error
initLog: mkdir(<logdir/info>) failed: error
<fname>: fopen(<logdir/file>) failed: error
getElogLock: Can't open existing lock file <logdir/file>: error
getElogLock: Error in opening lock file <logdir/file>: error
releaseElogLock: unlink(<logdir/lockfile>) failed: error
touchElogLock: Failed to open lock file <logdir/file>: error
touchElogLock: close <logdir/file> failed: error
```

mbatchd failed to create, remove, read, or write the log directory or a file in the log directory, for the reason given in *error*. Check that LSF administrator has read, write, and execute permissions on the `logdir` directory.

If `logdir` is on AFS, check that the instructions in the document “Installing LSF on AFS” on the Platform Web site have been followed. Use the `fs ls` command to verify that the LSF administrator owns `logdir` and that the directory has the correct `acl`.

```
replay_newjob: File <logfile> at line <line>: Queue <queue> not
found, saving to queue <lost_and_found>
```

```
replay_switchjob: File <logfile> at line <line>: Destination queue
<queue> not found, switching to queue <lost_and_found>
```

When `mbatchd` was reconfigured, jobs were found in *queue* but that *queue* is no longer in the configuration.

```
replay_startjob: JobId <jobId>: exec host <host> not found, saving
to host <lost_and_found>
```

When `mbatchd` was reconfigured, the event log contained jobs dispatched to *host*, but that *host* is no longer configured to be used by LSF.

```
do_restartReq: Failed to get hData of host <host_name>/<host_addr>
```

`mbatchd` received a request from `sbatchd` on host *host_name*, but that *host* is not known to `mbatchd`. Either the configuration file has been changed but `mbatchd` has not been reconfigured to pick up the new configuration, or *host_name* is a client host but the `sbatchd` daemon is running on that host. Run the following command to reconfigure the `mbatchd` or kill the `sbatchd` daemon on *host_name*.

```
badmin reconfig
```

LSF command messages

LSF daemon (LIM) not responding ... still trying

During LIM restart, LSF commands will fail and display this error message. User programs linked to the LIM API will also fail for the same reason. This message is displayed when LIM running on the master host list or server host list is restarted after configuration changes, such as adding new resources, binary upgrade, and so on.

Use `LSF_LIM_API_NTRIES` in `lsf.conf` or as an environment variable to define how many times LSF commands will retry to communicate with the LIM API while LIM is not available. `LSF_LIM_API_NTRIES` is ignored by LSF and EGO daemons and all EGO commands.

When `LSB_API_VERBOSE=Y` in `lsf.conf`, LSF batch commands will display the not responding retry error message to `stderr` when LIM is not available.

When `LSB_API_VERBOSE=N` in `lsf.conf`, LSF batch commands will not display the retry error message when LIM is not available.

Batch command client messages

LSF displays error messages when a batch command cannot communicate with `mbatchd`. The following table provides a list of possible error reasons and the associated error message output.

Point of failure	Possible reason	Error message output
Establishing a connection with <code>mbatchd</code>	<code>mbatchd</code> is too busy to accept new connections. The <code>connect()</code> system call times out.	LSF is processing your request. Please wait...
	<code>mbatchd</code> is down or there is no process listening at either the <code>LSB_MBD_PORT</code> or the <code>LSB_QUERY_PORT</code>	LSF is down. Please wait...
	<code>mbatchd</code> is down and the <code>LSB_QUERY_PORT</code> is busy	<code>bhosts</code> displays "LSF is down. Please wait..." <code>bjobs</code> displays "Cannot connect to LSF. Please wait..."
	Socket error on the client side	Cannot connect to LSF. Please wait...
	<code>connect()</code> system call fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...
Send/receive handshake message to/from <code>mbatchd</code>	<code>mbatchd</code> is busy. Client times out when waiting to receive a message from <code>mbatchd</code> .	LSF is processing your request. Please wait...
	Socket <code>read()/write()</code> fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...

EGO command messages

You cannot run the `egosh` command because the administrator has chosen not to enable EGO in `lsf.conf`: `LSF_ENABLE_EGO=N`.

If EGO is disabled, the `egosh` command cannot find `ego.conf` or cannot contact `vemkd` (not started).

Setting Daemon Message Log to Debug Level

The message log level for LSF daemons is set in `lsf.conf` with the parameter `LSF_LOG_MASK`. To include debugging messages, set `LSF_LOG_MASK` to one of:

- ◆ `LOG_DEBUG`
- ◆ `LOG_DEBUG1`
- ◆ `LOG_DEBUG2`
- ◆ `LOG_DEBUG3`

By default, `LSF_LOG_MASK=LOG_WARNING` and these debugging messages are not displayed.

The debugging log classes for LSF daemons is set in `lsf.conf` with the parameters `LSB_DEBUG_CMD`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, `LSB_DEBUG_SCH`, `LSF_DEBUG_LIM`, `LSF_DEBUG_RES`.

The location of log files is specified with the parameter `LSF_LOGDIR` in `lsf.conf`.

You can use the `lsadmin` and `badmin` commands to temporarily change the class, log file, or message log level for specific daemons such as `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd` without changing `lsf.conf`.

How the message log level takes effect

The message log level you set will only be in effect from the time you set it until you turn it off or the daemon stops running, whichever is sooner. If the daemon is restarted, its message log level is reset back to the value of `LSF_LOG_MASK` and the log file is stored in the directory specified by `LSF_LOGDIR`.

Limitations

When debug or timing level is set for `RES` with `lsadmin resdebug`, or `lsadmin restime`, the debug level only affects root `RES`. The root `RES` is the `RES` that runs under the root user ID.

Application `RES`s always use `lsf.conf` to set the debug environment. Application `RES`s are the `RES`s that have been created by `sbatchd` to service jobs and run under the ID of the user who submitted the job.

This means that any `RES` that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application `RES` will retain settings specified in `lsf.conf`.

Debug commands for daemons

The following commands set temporary message log level options for `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd`.

```
lsadmin limdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o [host_name]]
lsadmin resdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o [host_name]]
badmin mbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o]
badmin sbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o [host_name]]
badmin schddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o]
```

For a detailed description of `lsadmin` and `badmin`, see the *Platform LSF Command Reference*.

Examples

```
lsadmin limdebug -c "LC_MULTI LC_PIM" -f myfile hostA hostB
```

Log additional messages for the LIM daemon running on *hostA* and *hostB*, related to MultiCluster and PIM. Create log files in the `LSF_LOGDIR` directory with the name `myfile.lim.log.hostA`, and `myfile.lim.log.hostB`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

```
lsadmin limdebug -o hostA hostB
```

Turn off temporary debug settings for LIM on *hostA* and *hostB* and reset them to the daemon starting state. The message log level is reset back to the value of `LSF_LOG_MASK` and classes are reset to the value of `LSF_DEBUG_RES`, `LSF_DEBUG_LIM`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, and `LSB_DEBUG_SCH`. The log file is reset to the LSF system log file in the directory specified by `LSF_LOGDIR` in the format *daemon_name.log.host_name*.

```
badmin sbddebug -o
```

Turn off temporary debug settings for `sbatchd` on the local host (host from which the command was submitted) and reset them to the daemon starting state. The message log level is reset back to the value of `LSF_LOG_MASK` and classes are reset to the value of `LSF_DEBUG_RES`, `LSF_DEBUG_LIM`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, and `LSB_DEBUG_SCH`. The log file is reset to the LSF system log file in the directory specified by `LSF_LOGDIR` in the format *daemon_name.log.host_name*.

```
badmin mbddebug -l 1
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG1`. This command must be submitted from the host on which `mbatchd` is running because *host_name* cannot be specified with `mbddebug`.

```
badmin sbddebug -f hostB/myfolder/myfile hostA
```

Log messages for `sbatchd` running on *hostA*, to the directory `myfile` on the server *hostB*, with the file name `myfile.sbatchd.log.hostA`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

```
badmin schddebug -l 2
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG2`. This command must be submitted from the host on which `mbatchd` is running because *host_name* cannot be specified with `schddebug`.

```
badmin schddebug -l 1 -c "LC_PERFM"
```

```
badmin schdtime -l 2
```

Activate the LSF scheduling debug feature.

Log performance messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG`. Set the timing level for `mbschd` to include two levels of timing information.

```
lsadmin resdebug -o hostA
```

Turn off temporary debug settings for RES on *hostA* and reset them to the daemon starting state. The message log level is reset back to the value of `LSF_LOG_MASK` and classes are reset to the value of `LSF_DEBUG_RES`, `LSF_DEBUG_LIM`,

LSB_DEBUG_MBD, LSB_DEBUG_SBD, and LSB_DEBUG_SCH. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format *daemon_name.log.host_name*.

For timing level examples, see [Setting Daemon Timing Levels](#) on page 690.

Setting Daemon Timing Levels

The timing log level for LSF daemons is set in `lsf.conf` with the parameters `LSB_TIME_CMD`, `LSB_TIME_MBD`, `LSB_TIME_SBD`, `LSB_TIME_SCH`, `LSF_TIME_LIM`, `LSF_TIME_RES`.

The location of log files is specified with the parameter `LSF_LOGDIR` in `lsf.conf`. Timing is included in the same log files as messages.

To change the timing log level, you need to stop any running daemons, change `lsf.conf`, and then restart the daemons.

It is useful to track timing to evaluate the performance of the LSF system. You can use the `lsadmin` and `badmin` commands to temporarily change the timing log level for specific daemons such as `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd` without changing `lsf.conf`.

`LSF_TIME_RES` is not supported on Windows.

How the timing log level takes effect

The timing log level you set will only be in effect from the time you set it until you turn the timing log level off or the daemon stops running, whichever is sooner. If the daemon is restarted, its timing log level is reset back to the value of the corresponding parameter for the daemon (`LSB_TIME_MBD`, `LSB_TIME_SBD`, `LSF_TIME_LIM`, `LSF_TIME_RES`). Timing log messages are stored in the same file as other log messages in the directory specified with the parameter `LSF_LOGDIR` in `lsf.conf`.

Limitations

When debug or timing level is set for `RES` with `lsadmin resdebug`, or `lsadmin restime`, the debug level only affects root `RES`. The root `RES` is the `RES` that runs under the root user ID.

An application `RES` always uses `lsf.conf` to set the debug environment. An application `RES` is the `RES` that has been created by `sbatchd` to service jobs and run under the ID of the user who submitted the job.

This means that any `RES` that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application `RES` will retain settings specified in `lsf.conf`.

Timing level commands for daemons

The total execution time of a function in the LSF system is recorded to evaluate response time of jobs submitted locally or remotely.

The following commands set temporary timing options for `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd`.

```
lsadmin limtime [-l timing_level] [-f logfile_name] [-o] [host_name]
lsadmin restime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin mbdtime [-l timing_level] [-f logfile_name] [-o]
badmin sbdtime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin schdtime [-l timing_level] [-f logfile_name] [-o]
```

For debug level examples, see [Setting Daemon Message Log to Debug Level](#) on page 687.

For a detailed description of `lsadmin` and `badmin`, see the *Platform LSF Command Reference*.

Setting Daemon Timing Levels

Understanding Platform LSF Job Exit Information

Contents

- ◆ [Why did my job exit?](#) on page 694
- ◆ [How LSF translates events into exit codes](#) on page 694
- ◆ [Application and system exit values](#) on page 695
- ◆ [LSF job termination reason logging](#) on page 697
- ◆ [Job termination by LSF exit information](#) on page 701
- ◆ [LSF RMS integration exit values](#) on page 703

Why did my job exit?

LSF collects job information and reports the final status of a job. Traditionally jobs finishing normally report a status of 0, which usually means the job has finished normally. Any non-zero status means that the job has exited abnormally.

Most of the time, the abnormal job exit is related either to the job itself or to the system it ran on and not because of an LSF error. This document explains some of the information LSF provides about the abnormal job termination.

How LSF translates events into exit codes

The following table summarizes LSF exit behavior for some common error conditions.

Error condition	LSF exit code	Operating system	System exit code equivalent	Meaning
Command not found	127	all	1 or 127	Command shell returns 1 if command not found. If the command cannot be found inside a job script, LSF return exit code 127.
Directory not available for output	0	all	1	LSF sends the output back to user through email if directory not available for output (bsub -o).
LSF internal error	-127, 127	all	N/A	RES returns -127 or 127 for all internal problems.
Out of memory	N/A	all	N/A	Exit code depends on the error handling of the application itself.
LSF job states	0	all	N/A	Exit code 0 is returned for all job states

Host failure

If an LSF server host fails, jobs running on that host are lost. No other jobs are affected. Jobs can be submitted so that they are automatically rerun from the beginning or restarted from a checkpoint on another host if they are lost because of a host failure.

If all of the hosts in a cluster go down, all running jobs are lost. When a host comes back up and takes over as master, it reads the `lsb.events` file to get the state of all batch jobs. Jobs that were running when the systems went down are assumed to have exited, and email is sent to the submitting user. Pending jobs remain in their queues, and are scheduled as hosts become available.

Exited jobs

A job might terminate abnormally for various reasons. Job termination can happen from any state. An abnormally terminated job goes into EXIT state. The situations where a job terminates abnormally include:

- ◆ The job is cancelled by its owner or the LSF administrator while pending, or after being dispatched to a host.
- ◆ The job is not able to be dispatched before it reaches its termination deadline, and thus is aborted by LSF.

- ◆ The job fails to start successfully. For example, the wrong executable is specified by the user when the job is submitted.

The job exits with a non-zero exit status.

You can configure hosts so that LSF detects an abnormally high rate of job exit from a host. See *Administering Platform LSF* for more information.

Application and system exit values

LSF monitors a job while running and returns the exit code returned from the job itself. LSF collects this exit code via `wait3()` system call on UNIX platforms. The exit code is a result of the system exit values. Use `bhist` or `bjobs` to see the exit code for your job.

Application exit values

The most common cause of abnormal LSF job termination is due to application system exit values. If your application had an explicit exit value less than 128, `bjobs` and `bhist` display the actual exit code of the application; for example, `Exited with exit code 3`. You would have to refer to the application code for the meaning of exit code 3.

It is possible for a job to explicitly exit with an exit code greater than 128, which can be confused with the corresponding UNIX signal. Make sure that applications you write do not use exit codes greater than 128.

System signal exit values

When you send a signal that terminates the job, LSF reports either the signal or the `signal_value+128`. If the return status is greater than 128, and the job was terminated with a signal, then `return_status-128=signal`. For example, return status 133 means that the job was terminated with signal 5 (SIGTRAP on most systems, $133-128=5$). A job with exit status 130 was terminated with signal 2 (SIGINT on most systems, $130-128=2$).

Some operating systems define exit codes as 0-255. As a result, negative exit values or values > 255 may have a wrap-around effect on that range. The most common example of this is a program that exits -1 will be seen with "exit code 255" in LSF.

How or why the job may have been signaled, or exited with a certain exit code, can be application and/or system specific. The application or system logs might be able to give a better description of the problem.

TIP: Termination signals are operating system dependent, so signal 5 may not be SIGTRAP and 11 may not be SIGSEGV on all UNIX and Linux systems. You need to pay attention to the execution host type in order to correct translate the exit value if the job has been signaled.

bhist and bjobs output

In most cases, `bjobs` and `bhist` show the application exit value ($128 + \text{signal}$). In some cases, `bjobs` and `bhist` show the actual signal value.

If LSF sends catchable signals to the job, it displays the exit value. For example, if you run `bkill jobID` to kill the job, LSF passes SIGINT, which causes the job to exit with exit code 130 (SIGINT is 2 on most systems, $128+2=130$).

Application and system exit values

If LSF sends uncatchable signals to the job, then the entire process group for the job exits with the corresponding signal. For example, if you run `bkill -s SEGV jobID` to kill the job, `bjobs` and `bhist` show

```
Exited by signal 7
```

Example

The following example shows a job that exited with exit code 139, which means that the job was terminated with signal 11 (SIGSEGV on most systems, 139-128=11).

This means that the application had a core dump.

```
bjobs -l 2012
```

```
Job <2012>, User , Project , Status , Queue , Command
Fri Dec 27 22:47:28: Submitted from host , CWD <${HOME}>;
Fri Dec 27 22:47:37: Started on , Execution Home , Execution CWD ;
Fri Dec 27 22:48:02: Exited with exit code 139. The CPU time used is 0.2 seconds.
```

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

	cpuspeed	bandwidth
loadSched	-	-
loadStop	-	-

LSF job termination reason logging

When LSF takes action on a job, it may send multiple signals. In the case of job termination, LSF will send, SIGINT, SIGTERM and SIGKILL in succession until the job has terminated. As a result, the job may exit with any of those corresponding exit values at the system level. Other actions may send "warning" signals to applications (SIGUSR2) etc. For specific signal sequences, refer to the LSF documentation for that feature.

Run `bhist` to see the actions that LSF takes on a job:

```
bhist -l 1798
```

```
Job <1798>, User <user1>, Command <sleep 10000>
Tue Feb 25 16:35:31: Submitted from host <hostA>, to Queue <normal>, CWD <${HOME}/lsf_7.0/conf/lsbatch/lsf_7.0/configdir>;
Tue Feb 25 16:35:51: Dispatched to <hostA>;
Tue Feb 25 16:35:51: Starting (Pid 12955);
Tue Feb 25 16:35:53: Running with execution home </home/user1>, Execution CWD </home/user1/Testing/lsf_7.0/conf/lsbatch/lsf_7.0/configdir>, Execution Pid <12955>;
Tue Feb 25 16:38:20: Signal <KILL> requested by user or administrator <user1>;
Tue Feb 25 16:38:22: Exited with exit code 130. The CPU time used is 0.1 seconds;
```

```
Summary of time in seconds spent in various states by Tue Feb 25 16:38:22
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
20	0	151	0	0	0	171

Here we see that LSF itself sent the signal to terminate the job, and the job exits 130 (130-128 = 2 = SIGINT).

When a job finishes, LSF reports the last job termination action it took against the job and logs it into `lsb.acct`.

If a running job exits because of node failure, LSF sets the correct exit information in `lsb.acct`, `lsb.events`, and the job output file.

View logged job exit information (bacct -l)

1 Use bacct -l to view job exit information logged to lsb.acct:

```
bacct -l 7265
```

Accounting information about jobs that are:

- submitted by all users.
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.

```
-----
Job <7265>, User <lfsadmin>, Project <default>, Status <EXIT>, Queue <normal>,
      Command <srunk sleep 100000>
```

```
Thu Sep 16 15:22:09: Submitted from host <hostA>, CWD <${HOME}>;
```

```
Thu Sep 16 15:22:20: Dispatched to 4 Hosts/Processors <4*hostA>;
```

```
Thu Sep 16 15:23:21: Completed <exit>; TERM_RUNLIMIT: job killed after reaching
      LSF run time limit.
```

Accounting information about this job:

```
Share group charged </lfsadmin>
```

CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.04	11	72	exit	0.0006	OK	OK

```
-----
SUMMARY:      ( time unit: second )
```

Total number of done jobs:	0	Total number of exited jobs:	1
Total CPU time consumed:	0.0	Average CPU time consumed:	0.0
Maximum CPU time of a job:	0.0	Minimum CPU time of a job:	0.0
Total wait time in queues:	11.0		
Average wait time in queue:	11.0		
Maximum wait time in queue:	11.0	Minimum wait time in queue:	11.0
Average turnaround time:	72 (seconds/job)		
Maximum turnaround time:	72	Minimum turnaround time:	72
Average hog factor of a job:	0.00 (cpu time / turnaround time)		
Maximum hog factor of a job:	0.00	Minimum hog factor of a job:	0.00

Termination reasons displayed by bacct

When LSF detects that a job is terminated, `bacct -l` displays one of the following termination reasons:

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_ADMIN	Job killed by root or LSF administrator	15
TERM_BUCKET_KILL	Job killed with <code>bkill -b</code>	23
TERM_CHKPNT	Job killed after checkpointing	13
TERM_CPULIMIT	Job killed after reaching LSF CPU usage limit	12
TERM_CWD_NOTEXIST	Current working directory is not accessible or does not exist on the execution host	25
TERM_DEADLINE	Job killed after deadline expires	6
TERM_EXTERNAL_SIGNAL	Job killed by a signal external to LSF	17
TERM_FORCE_ADMIN	Job killed by root or LSF administrator without time for cleanup	9
TERM_FORCE_OWNER	Job killed by owner without time for cleanup	8
TERM_LOAD	Job killed after load exceeds threshold	3
TERM_MEMLIMIT	Job killed after reaching LSF memory usage limit	16
TERM_OTHER	Member of a chunk job in WAIT state killed and requeued after being switched to another queue.	4
TERM_OWNER	Job killed by owner	14
TERM_PREEMPT	Job killed after preemption	1
TERM_PROCESSLIMIT	Job killed after reaching LSF process limit	7
TERM_REQUEUE_ADMIN	Job killed and requeued by root or LSF administrator	11
TERM_REQUEUE_OWNER	Job killed and requeued by owner	10
TERM_RMS	Job exited from an RMS system error	18
TERM_RUNLIMIT	Job killed after reaching LSF run time limit	5
TERM_SLURM	Job terminated abnormally in SLURM (node failure)	22
TERM_SWAP	Job killed after reaching LSF swap usage limit	20
TERM_THREADLIMIT	Job killed after reaching LSF thread limit	21
TERM_UNKNOWN	LSF cannot determine a termination reason—0 is logged but TERM_UNKNOWN is not displayed	0
TERM_WINDOW	Job killed after queue run window closed	2
TERM_ZOMBIE	Job exited while LSF is not available	19

TIP: The integer values logged to the JOB_FINISH event in `lsb.acct` and termination reason keywords are mapped in `lsbatch.h`.

Restrictions

- ◆ If a queue-level `JOB_CONTROL` is configured, LSF cannot determine the result of the action. The termination reason only reflects what the termination reason could be in LSF.
- ◆ LSF cannot be guaranteed to catch any external signals sent directly to the job.

LSF job termination reason logging

- ◆ In MultiCluster, a `brequeue` request sent from the submission cluster is translated to `TERM_OWNER` or `TERM_ADMIN` in the remote execution cluster. The termination reason in the email notification sent from the execution cluster as well as that in the `lsb.acct` is set to `TERM_OWNER` or `TERM_ADMIN`.

Example output of `bacct` and `bhist`

Example termination cause	Termination reason in <code>bacct -l</code>	Example <code>bhist</code> output
<code>bkill -s KILL</code> <code>bkill job_ID</code>	Completed <exit>; <code>TERM_OWNER</code> or <code>TERM_ADMIN</code>	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
<code>bkill -r</code>	Completed <exit>; <code>TERM_FORCE_ADMIN</code> or <code>TERM_FORCE_OWNER</code> when <code>sbatchd</code> is not reachable. Otherwise, <code>TERM_USER</code> or <code>TERM_ADMIN</code>	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
<code>TERMINATE_WHEN</code>	Completed <exit>; <code>TERM_LOAD/</code> <code>TERM_WINDOWS/</code> <code>TERM_PREEMPT</code>	Thu Mar 13 17:33:16: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:33:18: Exited by signal 2. The CPU time used is 0.1 seconds;
Memory limit reached	Completed <exit>; <code>TERM_MEMLIMIT</code>	Thu Mar 13 19:31:13: Exited by signal 2. The CPU time used is 0.1 seconds;
Run limit reached	Completed <exit>; <code>TERM_RUNLIMIT</code>	Thu Mar 13 20:18:32: Exited by signal 2. The CPU time used is 0.1 seconds.
CPU limit	Completed <exit>; <code>TERM_CPULIMIT</code>	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Swap limit	Completed <exit>; <code>TERM_SWAPLIMIT</code>	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Regular job exits when host crashes	Rusage 0, Completed <exit>; <code>TERM_ZOMBIE</code>	Thu Jun 12 15:49:02: Unknown; unable to reach the execution host; Thu Jun 12 16:10:32: Running; Thu Jun 12 16:10:38: Exited with exit code 143. The CPU time used is 0.0 seconds;
<code>brequeue -r</code>	For each requeue, Completed <exit>; <code>TERM_REQUEUE_ADMIN</code> or <code>TERM_REQUEUE_OWNER</code>	Thu Mar 13 17:46:39: Signal <REQUEUE_PEND> requested by user or administrator <user2>; Thu Mar 13 17:46:56: Exited by signal 2. The CPU time used is 0.1 seconds;
<code>bchkpnt -k</code>	On the first run: Completed <exit>; <code>TERM_CHKPNT</code>	Wed Apr 16 16:00:48: Checkpoint succeeded (actpid 931249); Wed Apr 16 16:01:03: Exited with exit code 137. The CPU time used is 0.0 seconds;
<code>Kill -9 <RES></code> and job	Completed <exit>; <code>TERM_EXTERNAL_SIGNAL</code>	Thu Mar 13 17:30:43: Exited by signal 15. The CPU time used is 0.1 seconds;
Others	Completed <exit>;	Thu Mar 13 17:30:43: Exited with 3; The CPU time used is 0.1 seconds;

Job termination by LSF exit information

LSF also provides additional information in the POST_EXEC of the job. Use this information to detect conditions where LSF has terminated the job and take the appropriate action.

The job exit information in the POST_EXEC is defined in 2 parts:

- ◆ **LSB_JOBEXIT_STAT**—the raw `wait3()` output (converted using the wait macros `/usr/include/sys/wait.h`)
- ◆ **LSB_JOBEXIT_INFO**—defined only if the job exit was due to a defined LSF reason.

Queue-level POST_EXEC commands should be written by the cluster administrator to perform whatever task is necessary for specific exit situations.

TIP: System level enforced limits like CPU and Memory (listed above), cannot be shown in the **LSB_JOBEXIT_INFO** since it is the operating system performing the action and not LSF. Set appropriate parameters in the queue or at job submission to allow LSF to enforce the limits, which makes this information available to LSF.

Common LSB_JOBEXIT_STAT and LSB_JOBEXIT_INFO values

The following is a table of common scenarios covered and not covered by the **LSB_JOBEXIT_INFO**

Example termination cause	LSB_JOBEXIT_STAT	LSB_JOBEXIT_INFO	Example bhist output
Job killed with the SIGINT <code>bkill -s INT 520</code>	33280	SIGNAL 2 INT	Fri Feb 14 16:48:00: Exited with exit code 130. The CPU time used is 0.2 seconds;
Job killed with SIGTERM <code>bkill -s TERM 521</code>	36608	SIGNAL 15 TERM	Fri Feb 14 16:49:50: Exited with exit code 143. The CPU time used is 0.2 seconds;
Job killed with SIGKILL <code>bkill -s KILL 522</code>	33280	SIGNAL -14 SIG_TERM_USER	Fri Feb 14 16:51:03: Exited with exit code 130. The CPU time used is 0.2 seconds;
Automatic migration when MIG is defined at queue level	33280	SIGNAL -1 SIG_CHKPNT	Fri Feb 14 17:32:17: Job has been requeued; Fri Feb 14 17:32:17: Pending: Migrating job is waiting for rescheduling;
<code>bsub -l "hostname;exit 130"</code>	33280	Undefined	Fri Feb 14 14:41:51: Exited with exit code 130. The CPU time used is 0.2 seconds;
Killing the job with <code>bkill</code> command <code>bkill 210</code>	33280	SIGNAL -14 SIG_TERM_USER	Fri Feb 14 14:45:51: Exited with exit code 130. The CPU time used is 0.2 seconds;

Job termination by LSF exit information

Example termination cause	LSB_JOBEXIT_STAT	LSB_JOBEXIT_INFO	Example bhist output
Job being brequeued. brequeue -r Job <211> is being requeued	33280	SIGNAL -23 SIG_KILL_REQUEUE	Fri Feb 14 14:48:15: Signal <REQUEUE_PEND> requested by user or administrator <iayaz>; Fri Feb 14 14:48:18: Exited with exit code 130. The CPU time used is 0.2 second
Job being migrated bmig -m togni Job <213> is being migrated	33280	SIGNAL -1 SIG_CHKPNT	Fri Feb 14 15:04:42: Migration requested by user or administrator <iayaz>; Specified Hosts <togni>; Fri Feb 14 15:04:44: Job is being requeued; Fri Feb 14 15:05:01: Job has been requeued; Fri Feb 14 15:05:01: Pending: Migrating job is waiting for rescheduling;
Job killed due REQUEUE_EXIT_VALUE bsub "sleep 100;exit 34"	8704	Undefined	Fri Feb 14 15:10:21: Pending: Requeued job is waiting for rescheduling;(exit code 34)>;
Job killed by LSF when CPULIMIT enforced by LSF	158	SIGNAL -24 SIG_TERM_CPULIMIT	Wed Feb 19 14:18:13: Exited by signal 30. The CPU time used is 89.4 seconds.
Job killed because queue level CPULIMIT is reached.	40448	Undefined	Fri Feb 14 15:30:01: Exited with exit code 158. The CPU time used is 61.2 seconds;
Job killed because queue level RUNLIMIT is reached.	37120	Undefined	Fri Feb 14 15:37:44: Exited with exit code 145. The CPU time used is 0.2 seconds;
Job killed due to the check pointing. bchkpnt -k 838 Job <838> is being checkpointed	9	SIGNAL -1 SIG_CHKPNT	Fri Feb 14 17:59:12: Checkpoint succeeded (actpid 25298); Fri Feb 14 17:59:12: Exited by signal 9. The CPU time used is 0.1 seconds;
Job killed when reaches the MEMLIMIT bsub -M 5 "/home/iayaz/script/me mwrite -m 10 -r 2"	2	SIGNAL -25 SIG_TERM_MEMLIMIT	Fri Feb 21 10:50:50: Exited by signal 2. The CPU time used is 0.1 seconds;
Job killed when termination time approaches bsub -t 21:11:10 sleep 500;date	37120	Undefined	Exited with exit code 145. The CPU time used is 0.2 seconds;
Job killed when TERMINATE_WHEN = LOAD	33280	SIGNAL -15 SIG_TERM_LOAD	Exited with exit code 130. The CPU time used is 7.2 seconds.
Job killed when TERMINATE_WHEN = PREEMPT	33280	SIGNAL -16 SIG_TERM_PREEMPT	Exited with exit code 130. The CPU time used is 0.3 seconds;

LSF RMS integration exit values

For the RMS integrations with LSF (HP AlphaServer SC and Linux QsNet), LSF jobs running through RMS will return `rms_run()` return code as the job exit code. RMS documents certain exit codes and corresponding job exit reasons.

See the `rms_run()` man page for more information.

Upon successful completion, `rms_run()` returns the global OR of the exit status values of the processes in the parallel program. If one of the processes is killed, `rms_run()` returns a status value of 128 plus the signal number. It can also return the following codes:

Return Code	RMS Meaning
0	A process exited with the code 127 (GLOBAL EXIT), which indicates success, causing all of the processes to exit.
123	A process exited with the code 123 (GLOBAL ERROR) causing all of the processes to exit.
124	The node the job executing on has been removed from the system.
125	One or more processes were still running when the exit timeout expired.
126	The resource is inadequate for the request.

LSF RMS integration exit values

VIII

LSF Utilities

[Using lstcsh on page 707](#)

Using lstcsh

This chapter describes `lstcsh`, an extended version of the `tcsh` command interpreter. The `lstcsh` interpreter provides transparent load sharing of user jobs.

This chapter is not a general description of the `tcsh` shell. Only load sharing features are described in detail.

Interactive tasks, including `lstcsh`, are not supported on Windows.

Contents

- ◆ [About lstcsh](#) on page 707
- ◆ [Starting lstcsh](#) on page 710
- ◆ [Using lstcsh as Your Login Shell](#) on page 710
- ◆ [Host Redirection](#) on page 711
- ◆ [Task Control](#) on page 712
- ◆ [Built-in Commands](#) on page 712
- ◆ [Writing Shell Scripts in lstcsh](#) on page 714

About lstcsh

The `lstcsh` shell is a load-sharing version of the `tcsh` command interpreter. It is compatible with `csh` and supports many useful extensions. `csh` and `tcsh` users can use `lstcsh` to send jobs to other hosts in the cluster without needing to learn any new commands. You can run `lstcsh` from the command-line, or use the `chsh` command to set it as your login shell.

With `lstcsh`, your commands are sent transparently for execution on faster hosts to improve response time or you can run commands on remote hosts explicitly.

`lstcsh` provides a high degree of network transparency. Command lines executed on remote hosts behave the same as they do on the local host. The remote execution environment is designed to mirror the local one as closely as possible by using the same values for environment variables, terminal setup, current working directory, file creation mask, and so on. Each modification to the local set of environment

variables is automatically reflected on remote hosts. Note that shell variables, the nice value, and resource usage limits are not automatically propagated to remote hosts.

For more details on `lstcsh`, see the `lstcsh(1)` man page.

In this section

- ◆ [Task Lists](#) on page 708
- ◆ [Local and Remote Modes](#) on page 708
- ◆ [Automatic Remote Execution](#) on page 709

Task Lists

LSF maintains two task lists for each user, a local list (`.lsftask`) and a remote list (`lsf.task`). Commands in the local list must be executed locally. Commands in the remote list can be executed remotely.

See the *Platform LSF Configuration Reference* for information about the `.lsftask` and `lsf.task` files.

Changing task list membership

You can use the LSF commands `lsltasks` and `lsrtasks` to inspect and change the memberships of the local and remote task lists.

Task lists and resource requirements

Resource requirements for specific commands can be configured using task lists. You can optionally associate resource requirements with each command in the remote list to help LSF find a suitable execution host for the command.

If there are multiple eligible commands on a command-line, their resource requirements are combined for host selection.

If a command is in neither list, you can choose how `lstcsh` handles the command.

Local and Remote Modes

`lstcsh` has two modes of operation:

- ◆ Local
- ◆ Remote

Local mode

The local mode is the default mode. In local mode, a command line is eligible for remote execution only if all of the commands on the line are present in the remote task list, or if the `@` character is specified on the command-line to force it to be eligible.

See [@ character](#) on page 709 for more details.

Local mode is conservative and can fail to take advantage of the performance benefits and load-balancing advantages of LSF.

Remote mode

In remote mode, a command line is considered eligible for remote execution if none of the commands on the line is in the local task list.

Remote mode is aggressive and makes more extensive use of LSF. However, remote mode can cause inconvenience when `lstcsh` attempts to send host-specific commands to other hosts.

Automatic Remote Execution

Every time you enter a command, `lscsh` looks in your task lists to determine whether the command can be executed on a remote host and to find the configured resource requirements for the command.

See the *Platform LSF Configuration Reference* for information about task lists and `lsf.task` file.

If the command can be executed on a remote host, `lscsh` contacts LIM to find the best available host.

The first time a command is run on a remote host, a server shell is started on that host. The command is sent to the server shell, and the server shell starts the command on the remote host. All commands sent to the same host use the same server shell, so the start-up overhead is only incurred once.

If no host is found that meets the resource requirements of your command, the command is run on the local host.

Differences from Other Shells

When a command is running in the foreground on a remote host, all keyboard input (type-ahead) is sent to the remote host. If the remote command does not read the input, it is lost.

`lscsh` has no way of knowing whether the remote command reads its standard input. The only way to provide any input to the command is to send everything available on the standard input to the remote command in case the remote command needs it. As a result, any type-ahead entered while a remote command is running in the foreground, and not read by the remote command, is lost.

@ character

The @ character has a special meaning when it is preceded by white space. This means that the @ must be escaped with a backslash `\` to run commands with arguments that start with @, like `finger`. This is an example of using `finger` to get a list of users on another host:

```
finger @other.domain
```

Normally the `finger` command attempts to contact the named host. Under `lscsh`, the @ character is interpreted as a request for remote execution, so the shell tries to contact the RES on the host `other.domain` to remotely execute the `finger` command. If this host is not in your LSF cluster, the command fails. When the @ character is escaped, it is passed to `finger` unchanged and `finger` behaves as expected.

```
finger \@hostB
```

Limitations

A shell is a very complicated application by itself. `lscsh` has certain limitations:

Native language system

Native Language System is not supported. To use this feature of the `tcsh`, you must compile `tcsh` with `SHORT_STRINGS` defined. This causes complications for characters flowing across machines.

Starting lstcsh

Shell variables

Shell variables are not propagated across machines. When you set a shell variable locally, then run a command remotely, the remote shell will not see that shell variable. Only environment variables are automatically propagated.

fg command

The `fg` command for remote jobs must use `@`, as shown by examples in [Task Control](#) on page 712.

tcsh version

l`stcsh` is based on `tcsh 6.03` (7 bit mode). It does not support the new features of the latest `tcsh`.

Starting lstcsh

Start lstcsh

If you normally use some other shell, you can start `lstcsh` from the command-line.

-
- 1 Make sure that the LSF commands are in your `PATH` environment variable, then enter:

```
lstcsh
```

If you have a `.cshrc` file in your home directory, `lstcsh` reads it to set variables and aliases.

Exit lstcsh

-
- 1 Use the `exit` command to get out of `lstcsh`.
-

Using lstcsh as Your Login Shell

If your system administrator allows, you can use LSF as your login shell. The `/etc/shells` file contains a list of all the shells you are allowed to use as your login shell.

Set your login shell

Using `chsh`

The `chsh` command can set your login shell to any of those shells. If the `/etc/shells` file does not exist, you cannot set your login shell to `lstcsh`.

-
- 1 Run the command:

```
chsh user3 /usr/share/lsf/bin/lstcsh
```

The next time `user3` logs in, the login shell will be `lstcsh`.

Using a standard system shell

if you cannot set your login shell using `chsh`, you can use one of the standard system shells to start `lstcsh` when you log in.

To set up `lstcsh` to start when you log in:

- 1 Use `chsh` to set `/bin/sh` to be your login shell.
- 2 Edit the `.profile` file in your home directory to start `lstcsh`, as shown below:


```
SHELL=/usr/share/lsf/bin/lstcsh
export SHELL
exec $SHELL -l
```

Host Redirection

Host redirection overrides the task lists, so you can force commands from your local task list to execute on a remote host or override the resource requirements for a command.

You can explicitly specify the eligibility of a command-line for remote execution using the `@` character. It may be anywhere in the command line except in the first position (`@` as the first character on the line is used to set the value of shell variables).

You can restrict who can use `@` for host redirection in `lstcsh` with the parameter `LSF_SHELL_AT_USERS` in `lsf.conf`. See the *Platform LSF Configuration Reference* for more details.

Examples

```
hostname @hostD
<< remote execution on hostD >>
hostD

hostname @/type==linux
<< remote execution on hostB >>
hostB
```

@ character

<code>@</code>	<code>@</code> followed by nothing means that the command line is eligible for remote execution.
<code>@host_name</code>	<code>@</code> followed by a host name forces the command line to be executed on that host.
<code>@local</code>	<code>@</code> followed by the reserved word <code>local</code> forces the command line to be executed on the local host only.
<code>@/res_req</code>	<code>@</code> followed by <code>/</code> and a resource requirement string means that the command is eligible for remote execution and that the specified resource requirements must be used instead of those in the remote task list.

For ease of use, the host names and the reserved word `local` following `@` can all be abbreviated as long as they do not cause ambiguity.

Similarly, when specifying resource requirements following the @, it is necessary to use / only if the first requirement characters specified are also the first characters of a host name. You do not have to type in resource requirements for each command line you type if you put these task names into remote task list together with their resource requirements by running `lsrtasks`.

Task Control

Task control in `lstcsh` is the same as in `tcsh` except for remote background tasks. `lstcsh` numbers shell tasks separately for each execution host.

jobs command

The output of the built-in command `jobs` lists background tasks together with their execution hosts. This break of transparency is intentional to give you more control over your background tasks.

```
sleep 30 @hostD &
<< remote execution on hostD >>
[1] 27568
sleep 40 @hostD &
<< remote execution on hostD >>
[2] 10280
sleep 60 @hostB &
<< remote execution on hostB >>
[1] 3748
jobs
<hostD>
[1] + Running          sleep 30
[2]   Running          sleep 40
<hostB>
[1] + Running          sleep 60
```

Bring a remote background task to the foreground

- 1 To bring a remote background task to the foreground, the host name must be specified together with @, as in the following example:

```
fg %2 @hostD
<< remote execution on hostD >>
sleep 40
```

Built-in Commands

`lstcsh` supports two built-in commands to control load sharing, `lsmode` and `connect`.

In this section

- ◆ [lsmode](#) on page 713
- ◆ [connect](#) on page 714

lsmode

Syntax

```
lsmode [on|off] [local|remote] [e|-e] [v|-v] [t|-t]
```

Description

The `lsmode` command reports that LSF is enabled if `lstcsh` was able to contact LIM when it started up. If LSF is disabled, no load-sharing features are available.

The `lsmode` command takes a number of arguments that control how `lstcsh` behaves.

With no arguments, `lsmode` displays the current settings:

```
lsmode
LSF
Copyright Platform Computing Corporation
LSF enabled, local mode, LSF on, verbose, no_eligibility_verbose,
no timing.
```

Options

[on | off]

Turns load sharing on or off. When turned off, you can send a command line to a remote host only if force eligibility is specified with `@`.

The default is on.

[local | remote]

Sets `lstcsh` to use local or remote mode.

The default is local. See [Local and Remote Modes](#) on page 708 for a description of local and remote modes.

[e | -e]

Turns eligibility verbose mode on (e) or off (-e). If eligibility verbose mode is on, `lstcsh` shows whether the command is eligible for remote execution, and displays the resource requirement used if the command is eligible.

The default is off.

[v | -v]

Turns task placement verbose mode on (v) or off (-v). If verbose mode is on, `lstcsh` displays the name of the host on which the command is run, if the command is not run on the local host. The default is on.

[t | -t]

Turns wall-clock timing on (t) or off (-t).

If timing is on, the actual response time of the command is displayed. This is the total elapsed time in seconds from the time you submit the command to the time the prompt comes back.

This time includes all remote execution overhead. The `csch time` builtin does not include the remote execution overhead.

This is an impartial way of comparing the response time of jobs submitted locally or remotely, because all the load sharing overhead is included in the displayed elapsed time.

The default is off.

connect

Syntax `connect [host_name]`

Description `lstcsh` opens a connection to a remote host when the first command is executed remotely on that host. The same connection is used for all future remote executions on that host.

The `connect` command with no argument displays connections that are currently open.

The `connect host_name` command creates a connection to the named host. By connecting to a host before any command is run, the response time is reduced for the first remote command sent to that host.

`lstcsh` has a limited number of ports available to connect to other hosts. By default each shell can only connect to 15 other hosts.

Examples

```
connect
CONNECTED WITH          SERVER SHELL
hostA                    +

connect hostB
Connected to hostB

connect
CONNECTED WITH          SERVER SHELL
hostA                    +
hostB                    -
```

In this example, the `connect` command created a connection to host `hostB`, but the server shell has not started.

Writing Shell Scripts in lstcsh

You should write shell scripts in `/bin/sh` and use the `lstools` commands for load sharing. However, `lstcsh` can be used to write load-sharing shell scripts.

By default, an `lstcsh` script is executed as a normal `tosh` script with load-sharing disabled.

Run a script with load sharing enabled

The `lstcsh -L` option tells `lstcsh` that a script should be executed with load sharing enabled, so individual commands in the script may be executed on other hosts.

There are three different ways to run an `lstcsh` script with load sharing enabled:

- ◆ Run `lstcsh -L script_name`, or
- ◆ Make the script executable and put the following as the first line of the script. By default, `lstcsh` is installed in `LSF_BINDIR`.

The following assumes you installed `lstcsh` in the `/usr/share/lsf/bin` directory):

```
#!/usr/share/lsf/bin/lstcsh -L
```

- 1 Start an interactive `lstcsh`.
 - 2 Enable load sharing, and set to remote mode:
`lsmode on remote`
 - 3 Use the `source` command to read the script in.
-

Index

Symbols

- ! (NOT) operator, job dependencies 424
- # of Inst 192
- %I substitution string in job arrays 462
- %J substitution string in job arrays 462
- %USRCMD string in job starters 540
- && (AND) operator, job dependencies 424
- .cshrc file and lscsh 710
- .lsbatch directory 32
- .rhosts file
 - disadvantages 643
 - file transfer with lsrcp 659
 - host authentication 643
 - troubleshooting 674
- /etc/hosts file
 - example host entries 80
 - host naming 77
 - name lookup 79
 - troubleshooting 674
- /etc/hosts.equiv file
 - host authentication 642
 - troubleshooting 674
 - using rcp 659
- /etc/services file, adding LSF entries to 76
- /etc/shells file, and lscsh 710
- /etc/syslog.conf file 201, 663
- /usr/include/sys/syslog.h file 663
- @ (at sign) in lscsh 711
- || (OR) operator, job dependencies 424
- ~ (tilde)
 - not operator
 - host partition fairshare 302
 - host-based resources 252

A

- abnormal job termination 105, 694
- ABS_RUNLIMIT parameter in lsb.params 519
- absolute job priority scheduling
 - admin value 435
 - description 431
 - LSF feature interactions 439
 - modifying calculated APS value 435
 - priority factors 432
- absolute run time limit 519

- access permissions for interactive tasks 577
- account information, modify 185
- accounting information for advance reservations 416
- adaptive dispatch. *See* chunk jobs
- admin APS value 435
- administrator comments
 - logging in lsb.events
 - for host open and close 62
 - for mbatchd restart 48
 - for queue events 95
- ADMINISTRATORS, lsb.queues file 434
- administrators
 - cluster administrator description 45
 - primary LSF administrator 45
- ADMINISTRATORS parameter in lsf.cluster.cluster_name 45
- advance reservation
 - accounting information 416
 - adding and removing 407
 - changing job termination time (bmod -t) 414
 - commands 407
 - configuring user policies 405
 - description 404, 405
 - license 405
 - reservation ID 413
 - schmod_advrsv plugin 405
 - submitting jobs 414
 - user policies 405
 - viewing 410
 - viewing accounting information 416
 - weekly planner (brsvs -p) 411
- advanced dependency conditions 426
- AFS (Andrew File System)
 - overview 655
 - tokens for esub and eexec 544
- aliases
 - for resource names 284
 - host name ranges 78
 - using as host names 77
- allocation limits. *See* resource allocation limits
- AND operator (&&), job dependencies 424
- Andrew File System. *See* AFS
- application profiles

Index

- adding and removing 358
 - configuring, for chunk jobs 454
 - controlling jobs 359
 - default application profile 358
 - description 357
 - modifying jobs (bmod -app) 359
 - submitting jobs (bsub -app) 359
 - viewing
 - detailed information (bapp -l) 360
 - jobs (bjobs -app) 361
 - summary information (bacct -app) 361
 - summary information (bapp) 360
 - application-level, pre- and post-execution commands, configuring 532
 - APS. *See* absolute priority scheduling
 - APS_PRIORITY parameter in lsb.queues 434
 - architecture, EGO 171
 - architecture, viewing for hosts 58
 - arguments
 - passed to the LSF event program 624
 - passing to job arrays 462
 - at sign (@) in lscsh 711
 - augmentstarter job starter 541
 - authentication, security 642
 - automatic
 - duplicate event logging 665
 - event log archiving 665
 - job requeue 441
 - job rerun 444
 - queue selection 29
 - remote execution in lscsh 709
 - automatic time-based configuration, description 271
 - automount command, NFS (Network File System) 655, 672
 - automount option, /net 656
- ## B
- bacct -app 361
 - bacct command
 - CPU time display 521
 - SLA scheduling 334
 - bacct -U, advance reservations 416
 - backfill scheduling
 - default run limit 514
 - description 483
 - interruptible backfill 488
 - background jobs, bringing to foreground 712
 - badadmin command
 - hopen 62
 - hrestart 46
 - hshutdown 46
 - hstartup 46
 - logging administrator comments
 - for host open and close 62
 - for mbatchd restart 48
 - for queue events 95
 - LSF event logs 663
 - mbdrestart 46, 52
 - qact 95
 - qclose 95
 - qinact 95
 - qopen 95
- banded licensing 137
 - bapp 360
 - batch jobs
 - accessing files 655, 656
 - allocating processors 470
 - email about jobs
 - disabling 650
 - options 649
 - file access 543, 656
 - input and output 649
 - killing 112
 - pre- and post-execution commands 529
 - requeue 441
 - rerunning and restarting 444
 - signalling 112
 - batch log files. *See* log files
 - batch queues. *See* queues
 - bbot command 109
 - B-Class LSF license type 137
 - benchmarks for setting CPU factors 87
 - Berkeley Internet Name Domain (BIND), host naming 77
 - bgadd command, job group limits 123
 - bgdel command 122
 - bgmod command, job group limits 123
 - bhist command
 - job exit codes 668
 - LSF event logs 663
 - bhist -l 121
 - administrator changes to absolute job priority scheduling 436
 - bhosts command, checking time-based configuration 273
 - bhosts -l 54
 - bhosts -x, viewing host exception status 61
 - BIND (Berkeley Internet Name Domain), host naming 77
 - bjgroup command 119
 - SLA scheduling 340
 - bjobs -app 361
 - bjobs -aps
 - changes to absolute job priority scheduling 435
 - order of absolute job priority scheduling 437
 - bjobs command
 - reservation ID for advance reservation 413
 - SLA scheduling 334

- bjobs -g 120
 - bjobs -l, modified absolute job priority scheduling values 436
 - bjobs -x, viewing job exception status 108
 - bkill -app 359
 - bkill -g 122
 - black hole hosts 88, 124
 - bladmin chkconfig command, checking time-based configuration 274
 - bldloader 604
 - blimits -c command, checking time-based configuration 274
 - blimits command 384
 - blinfo command, checking time-based configuration 274
 - blstat command, checking time-based configuration 274
 - bmod, absolute job priority scheduling 435
 - bmod -app 359
 - bmod -g 121
 - bmod -is 654
 - bmod -t 414
 - bmod -Zs 654
 - Boolean resources 232
 - bparams command
 - checking time-based configuration 274
 - viewing configuration parameters 38
 - bqueues command
 - checking time-based configuration 274
 - cross-queue fairshare information 303
 - bqueues -l, absolute job priority scheduling 438
 - bresources command, checking time-based configuration 274
 - bresume -app 359
 - bresume -g 121
 - brsvadd -b, specifying begin times 408
 - brsvadd command 407
 - brsvadd -e, specifying end times 408
 - brsvadd -m, specifying a host list 407
 - brsvadd -R, specifying a resource requirement string 407
 - brsvadd -t, specifying recurring reservations 409
 - brsvdel command 410
 - brsvs command 410
 - brun command
 - advance reservation 416
 - forcing a job to run 111
 - job preemption 277
 - bsla command 332
 - EGO-enabled SLA scheduling 346
 - bstop -app 359
 - bstop -g 120
 - bsub -app 359
 - bsub command
 - email job notification 649
 - input and output 649
 - remote file access 657
 - submitting a job
 - associated to a job group 118
 - associated to a service class 329
 - bsub -f 657
 - bsub -is 652
 - bsub -sla 329
 - bsub -Zs 653
 - built-in load indices, overriding 258
 - built-in resources 232
 - busers command, checking time-based configuration 274
 - busy host status
 - lsload command 55
 - status load index 237
 - busy thresholds, tuning 628
- ## C
- calculating license key check sums (lmcksum) 153
 - calculation of required licenses 141
 - candidate master hosts, specifying 631
 - ceiling resource usage limit 513
 - ceiling run limit 513
 - chargeback fairshare 324
 - check_license script, for counted software licenses 262
 - checking
 - license server status (lmstat) 151
 - LSF floating client 164
 - checkout of licenses 141
 - checkpoint and restart, executables 449
 - checkpointable jobs, chunk jobs 457
 - checksum, calculating for license key (lmcksum) 153
 - chsh and lscsh 710
 - chunk jobs
 - absolute job priority scheduling 440
 - checkpointing 457
 - CHUNK_JOB_DURATION parameter in lsb.params 454
 - configuring application profile for 454
 - configuring queue for 454
 - description 453
 - fairshare scheduling 457
 - job controls 456
 - killing 456
 - limitations on queues 455
 - migrating 457
 - modifying 457
 - rerunnable 457
 - resource usage limits 511
 - resuming 457
 - submitting and controlling 456

Index

- WAIT status and pending reason 456
- CHUNK_JOB_DURATION, parameter in `lsb.params` 454
- CLEAN_PERIOD parameter in `lsb.params` 113
- client host, floating licenses 160
- closed host status 54
 - bhosts command 54
 - bhosts -l 54, 57
- closed_Adm condition, output of bhosts -l 54
- closed_Busy condition, output of bhosts -l 54
- closed_EGO condition, output of bhosts -l 54
- closed_Excl condition, output of bhosts -l 54
- closed_Full condition, output of bhosts -l 54
- closed_LIM condition, output of bhosts -l 54
- closed_Lock condition, output of bhosts -l 54
- closed_Wind condition, output of bhosts -l 54
- cluster administrators
 - description 45
 - role, description 199
 - viewing 38
- cluster names, viewing 38
- clusters
 - configuration file quick reference 51
 - protecting with strict checking 71
 - reconfiguring
 - commands 51
 - how reconfiguration affects licenses 52
 - viewing, information about 38
 - viewing information 38
- command file spooling
 - See also job file spooling
 - default directory 653
 - description 652
 - JOB_SPOOL_DIR parameter in `lsb.params` 652
- commands
 - built-in 713
 - checking time-based configuration 273
 - FLEXlm utilities 151
 - job starters 537
 - lmcksum (FLEXlm) 153
 - lmdown (FLEXlm) 153
 - lmhostid (FLEXlm) 153
 - lmremove (FLEXlm) 153
 - lmreread (FLEXlm) 153
 - lmstat (FLEXlm)
 - description 153
 - displaying license server status 151
 - example 152
 - using 151
 - lmver (FLEXlm) 153
 - lshosts -l 156
 - post-execution. See post-execution commands
 - pre-execution. See pre-execution commands
 - running under user ID 534
 - using in job control actions 556
- components, EGO 171
- Comprehensive System Accounting (IRIX CSA), configuring 662
- concurrent threads 519
- CONDENSE keyword in `lsb.hosts` 85
- CONDENSE_PENDING_REASONS parameter in `lsb.params` 588
- condensed host groups
 - defining 85
 - viewing 57
- configuration
 - adding and removing
 - application profiles 358
 - queues 98
 - commands for checking 273
 - default application profile 358
 - preselecting master hosts 630
 - removing, hosts 66
 - tuning
 - busy thresholds 628
 - LIM policies 628
 - load indices 629
 - load thresholds 629
 - mbatchd on UNIX 635
 - run windows 628
 - viewing, errors 52
- configuration files
 - location 131
 - non-shared file systems 656
 - reconfiguration quick reference 51
- configuration parameters. See individual parameter names
- consumer administrators
 - adding 199
 - role, description 199
- CONSUMER parameter in `lsb.serviceclasses` 345
- consumer users
 - adding 200
 - role, description 199
- consumers
 - about 172
 - creating 189
 - default 189
- CONTROL_ACTION parameter in `lsb.serviceclasses` 339
- core file size limit 515
- CORELIMIT parameter in `lsb.queues` 515
- counted software licenses
 - configuring 262
 - description 261
- CPU
 - factors
 - static resource 240

- time normalization 521
 - tuning in `lsf.shared` 86
 - limits
 - per job 516
 - per process 516
 - normalization 521
 - run queue length, description 566
 - time
 - cumulative and decayed 299
 - in dynamic user priority calculation 299
 - time limit, job-level resource limit 515
 - tuning CPU factors in `lsf.shared` 86
 - utilization, `ut` load index 238
 - viewing run queue length 87
 - CPU factor, non-normalized run time limit 519
 - CPU factor (`cpuf`) static resource 240
 - CPU time, idle job exceptions 100, 124
 - CPU time normalization 516
 - CPU utilization, optimization guide 186
 - `CPU_TIME_FACTOR` parameter in `lsb.params`, fairshare dynamic user priority 299
 - `cpuf` static resource 240
 - `CPULIMIT` parameter in `lsb.queues` 515
 - Cray, UNICOS accounting log files 662
 - cross-queue fairshare 302
 - CSA (IRIX Comprehensive System Accounting), configuring 662
 - `.cshrc` file and `lscsh` 710
 - cumulative CPU time 299
 - custom event handlers 624
 - custom file transfer, configuring 659
 - custom resources
 - adding 250
 - configuring 250
 - description 249
 - resource types 232
- D**
- DAEMON line, `license.dat` file 139
 - DAEMON line, editing 144
 - daemons
 - commands 46
 - debug commands 687
 - error logs 201, 662
 - restarting
 - `mbatchd` 48
 - `sbatchd` 47
 - shutting down
 - `mbatchd` 49
 - `sbatchd` 47
 - TCP service ports 75
 - `ypbind` 79
 - dashboard, about 186
 - data loader plug-ins 607
 - EGO 605
 - `egoconsumerresloader` 605
 - `egodynamicresloader` 606
 - `egoeventsloader` 606
 - `egostaticresloader` 606
 - LSF 604
 - `bldloader` 604
 - `desktopactiveeventloader` 605
 - `desktopclientloader` 605
 - `desktopjobloader` 604
 - `lsfbhostsloader` 605
 - `lsfeventsloader` 605
 - `lsfslloader` 605
 - `sharedresproploader` 605
 - `sharedresusagelower` 605
 - data purger 606, 608
 - record expiry time 615
 - schedule 608, 613
 - data segment size limit 516
 - `DATALIMIT` parameter in `lsb.queues` 516
 - date of expiry (demo) 136
 - DCE/DFS (Distributed File System)
 - credentials, `esub` and `eexec` 544
 - overview 655
 - deadline constraint scheduling
 - description 275
 - parallel jobs 493
 - deadlock, avoiding signal and job action 557
 - debug information
 - logging classes 203
 - logging levels 203
 - debug level
 - commands for daemons 687
 - setting temporarily 687
 - debug log classes, description 203
 - debug log levels, description 203
 - decayed
 - CPU time 299
 - run time 320
 - dedicated resources. *See* exclusive resources
 - DEFAULT, model or type with `lshosts` command 678
 - default
 - input file spooling 653
 - job control actions 553
 - `JOB_SPOOL_DIR` 653
 - LSF log file location 201, 661
 - `LSF_LOGDIR` 201, 663
 - output file spooling 653
 - queue, viewing 28
 - resource usage limits 513
 - run limit, backfill scheduling 514
 - UNIX directory structure 43
 - Windows directory structure 44
 - default normalization host 516

Index

- default SLA for EGO-enabled SLA 344
 - DEFAULT_APPLICATION parameter in lsb.params 358
 - DEFAULT_HOST_SPEC parameter
 - in lsb.params 516
 - in lsb.queues 521
 - DEFAULT_JOBGROUP parameter in lsb.params 115
 - DEFAULT_SLA_VELOCITY parameter in lsb.params 351
 - defined keyword 286
 - definitions
 - EGO 170
 - service instances 194
 - delayed SLA scheduling goals, control action 339
 - demand, defining in SDK 171
 - demo license 136
 - Depend, service property 193
 - dependency conditions
 - job arrays, operators 463
 - relational operators 425
 - dependency conditions. *See* job dependency conditions
 - dependency expressions, multiple conditions 424
 - desktopactiveeventloader 605
 - desktopclientloader 605
 - desktopjobloader 604
 - DFS (Distributed File System). *See* DCE/DFS
 - directories
 - default UNIX directory structure 43
 - default Windows directory structure 44
 - log, permissions and ownership 202, 662
 - .lsbatch 32
 - LSF_SERVERDIR, esub and eexec 544
 - remote access 543, 656
 - shared 32
 - user accounts 32
 - directory for license (demo) 143
 - directory for license (permanent) 146
 - disks, I/O rate 239
 - dispatch order, fairshare 300
 - dispatch turn, description 30
 - dispatch windows
 - description 420
 - hosts 62
 - queues 96
 - tuning for LIM 628
 - dispatch, adaptive. *See* chunk jobs
 - DISPATCH_WINDOW, queues 97
 - displaying
 - FLEXlm version information (lmver) 153
 - hardware host ID (lmhostid) 153
 - license server status (lmstat) 151, 153
 - licensed products 156
 - distributed license server hosts 154
 - distribution (partial licensing) 156
 - distribution of licenses 141
 - Domain Name Service (DNS), host naming 77
 - done job dependency condition 425
 - DONE job state
 - description 103
 - post-execution commands 531
 - done jobs, viewing 104
 - dual-core processor support 136
 - dual-stack hosts
 - defining official host name 81
 - dns configuration 81
 - duplicate event logging 665
 - after network partitioning 664
 - automatic 665
 - description 664
 - mbatchd restart with MAX_INFO_DIRS 590
 - dynamic
 - hosts, protecting with strict checking 71
 - master host 33
 - resources 232
 - user priority
 - description 298
 - formula 299
- ## E
- eadmin script
 - default exception actions 124
 - host and job exception handling 124
 - EADMIN_TRIGGER_DURATION parameter in lsb.params 101
 - E-Class LSF license type 137
 - eexec (external executable) script
 - description 543
 - passing data to execution environments 552
 - running as non-root 552
 - effective run queue length
 - built-in resources 238
 - description 566
 - tuning LIM 630
 - EGO
 - components 171
 - how it works 171
 - what it is 170
 - ego.conf file
 - EGO_LOG_MASK parameter 663
 - managing error logs 663
 - EGO_LOG_MASK parameter in ego.conf 663
 - EGO_RES_REQ parameter in lsb.serviceclasses 351
 - egoconsumerresloader 605
 - egodynamicresloader 606
 - EGO-enabled SLA scheduling
 - bsla command 346
 - configuring 344

- description 341
- egoeventsloader 606
- egostaticresloader 606
- egroup executable
 - external host groups 86
 - external user groups 133
- electronic mail. *See* email
- eligible hosts, viewing 31
- eligible hosts, viewing 31
- ELIM (external LIM)
 - counted software licenses 262
 - custom resources 254
 - debugging 258
- email
 - disabling batch job notification 650
 - job options 649
 - limiting the size of job email 650
- embedded submission options for interactive jobs 570
- ENABLE_DEFAULT_EGO_SLA parameter in `lsb.params` 344
- ENABLE_EXIT_RATE_PER_SLOT parameter in `lsb.params` 127
- ended job dependency condition 425
- environment, setting 188
- environment of a job 32
- environment variables. *See* individual environment variable names
- equal share fairshare 325
- error logs
 - EGO_LOG_MASK parameter 663
 - log directory, `LSF_LOGDIR` 201, 663
 - log files 201, 662
 - LSF daemons 201, 662
 - LSF_LOG_MASK parameter 663
 - managing log files 662
 - on UNIX and Windows 201, 663
- error messages, service 195
- errors, viewing in reconfiguration 52
- esub (external submission) executable
 - description 544
 - environment variables 544
 - job submission parameters 548
 - mandatory method (`LSB_ESUB_METHOD`) 550
 - pass data to execution environments 552
- esub method (`LSB_ESUB_METHOD`) 550
- `/etc/hosts` file
 - example host entries 80
 - host naming 77
 - name lookup 79
 - troubleshooting 674
- `/etc/hosts.equiv` file
 - host authentication 642
 - troubleshooting 674
- using `rcp` 659
- `/etc/services` file, adding LSF entries to 76
- `/etc/syslog.conf` file 201, 663
- evaluation license 136
- event generation 623
- event log archiving, automatic 665
- event log replication. *See* duplicate event logging
- event logging, `mbatchd` restart with `MAX_INFO_DIRS` 590
- event logs 609
 - automatic archiving 609, 665
 - configuring duplicate logging 665
 - duplicate logging 665
 - `ego.stream` file 609
 - logging administrator comments
 - for host open and close 62
 - for `mbatchd` restart 48
 - for queue events 95
 - `lsb.events` file 33
 - `lsb.stream` file 609
 - LSF Batch log file in `lsb.events` file 663
 - update interval 665
- Event Viewer, Windows 623
- `EVENT_UPDATE_INTERVAL` in `lsb.params` 665
- events
 - custom programs to handle 624
 - generated by LSF 624
 - logging 33
- example
 - demo license 139
 - partial licensing 157
 - permanent license 140
- `example.services` file 76
- examples
 - `/etc/hosts` file entries 80
 - `lstat` command 152
- exception handling
 - configuring host exceptions 88
 - configuring in queues 100
 - description 34
- exception status
 - for hosts, viewing with `bhosts` 60
 - for jobs
 - viewing with `bjobs` 108
 - viewing with `bqueues` 94
- exclusive jobs, requeue 442
- exclusive resources host-based resources, exclusive resources 286
- execution
 - environment 32
 - forcing for jobs 111
 - priority 240
- execution host, mandatory for parallel jobs 473
- exit codes, returned by jobs 668

Index

- exit dependency condition, relational operators 425
- exit job dependency condition 425
- EXIT job state
 - abnormal job termination 105, 694
 - pre- and post-execution commands 531
- exit rate for jobs 88, 124
- EXIT_RATE, bhosts -l 60
- EXIT_RATE parameter in lsb.hosts 88
- expiry date (demo) 136
- expiry time for mbatchd 636
- external
 - job dependency condition 425
 - job submission executable (esub) 544
 - LIM. *See* ELIM (external LIM)
 - load indices, using ELIM 254
 - user groups, configuring 133
- F**
- factor grace period, absolute job priority scheduling 433
- factor limit, absolute job priority scheduling 432
- factor weight, absolute job priority scheduling 432
- FAIRSHARE parameter in lsb.queues 304
- fairshare scheduling
 - absolute job priority scheduling 439
 - across queues 302
 - advance reservation 415
 - chargeback 324
 - chunk jobs 457
 - defining policies that apply to several queues 302
 - description 296
 - dynamic user priority
 - description 298
 - formula 299
 - equal share 325
 - global 324
 - hierarchical share tree 308
 - overview 295
 - parallel jobs 492
 - policies 296
 - priority user 325
 - resource usage measurement 298
 - static priority 326
 - user share assignment 297
 - viewing cross-queue fairshare information 303
- FAIRSHARE_QUEUES parameter
 - in bqueues 303
 - in lsb.queues 304
 - OBSOLETE 436
- fast job dispatching 585
- fault tolerance
 - description 33
 - non-shared file systems 656
- FCFS (first-come, first-served) scheduling 29
- FEATURE line
 - license.dat file (demo) 138
 - license.dat file (permanent) 139
- features (LSF) 147
- file (for demo license) 138
- file (for permanent license) 139
- file access, interactive tasks 577
- file preparation, job arrays 461
- file sharing 32
- file size usage limit 517
- file spooling. *See* command file spooling, job file spooling
- file systems
 - AFS (Andrew File System) 655
 - DCE/DFS (Distributed File System) 655
 - NFS (Network File System) 655
 - supported by LSF 655
- file transfer, lscrcp command 658
- file, updating 149
- FILELIMIT parameter in lsb.queues 517
- files
 - /etc/hosts
 - example host entries 80
 - host naming 77
 - name lookup 79
 - /etc/services, adding LSF entries to 76
 - automatic time-based configuration 271
 - copying across hosts 577
 - enabling utmp registration 571
 - hosts, configuring 80
 - if-else constructs 271
 - license.dat, location 146
 - lsb.params
 - CHUNK_JOB_DURATION parameter 454
 - JOB_ACCEPT_INTERVAL parameter 30
 - lsf.cluster.cluster_name,
 - FLOAT_CLIENTS_ADDR_RANGE parameter 163
 - lsf.conf
 - configuring TCP service ports 75
 - daemon service ports 75
 - lsf.shared, adding a custom host types and models 74
 - redirecting 563
 - redirecting stdout and stderr 577
 - resolv.conf 79
 - spooling command and job files 569
 - svc.conf (name services) 78
- filter settings, using 187
- finger command in lscsh 709
- first-come, first-served (FCFS) scheduling 29
- FLEXlm
 - calculating license key check sums

- (lmcksum) 153
- displaying
 - hardware host ID (lmhostid) 153
 - license server status (lmstat) 153
 - version information (lmver) 153
- removing a licensed feature (lmremove) 153
- rereading license file (lmreread) 153
- shutting down license server (lmdown) 153
- utility commands 151
- FLEXlm log file 152
- FLEXlm-based license 136
- floating client
 - description 160
 - enabling 161
 - resetting 160
 - specifying host or model type 161
 - verifying 164
- floating client licenses
 - FLOAT_CLIENTS parameter in
 - lsf.cluster.cluster_name 162
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsf.cluster.cluster_name 163
- floating LSF client
 - description 160
 - enabling 161
 - resetting 160
 - specifying host or model type 161
 - verifying 164
- floating LSF client licenses
 - FLOAT_CLIENTS parameter in
 - lsf.cluster.cluster_name 162
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsf.cluster.cluster_name 163
- floating software licenses
 - configuring dedicated queue for 264
 - managing with LSF 262
- forcing job execution 111
- formula, fairshare dynamic user priority calculation 299
- free memory 239
- FS absolute job priority scheduling factor 434
- G**
- gethostbyname function (host naming) 79
- global fairshare 324
- GLOBAL_EXIT_RATE parameter in lsb.params 127
- GLOBEtrotter Software 140
- goal-oriented scheduling. *See* SLA scheduling
- goals, SLA scheduling 328
- grace period 141
 - absolute job priority scheduling factor 433
- groups
 - external host 86
 - external user groups 133
- hosts 83
 - users 131
- groups, specifying 323
- GUI (see "Management Console")
- H**
- hard resource limits, description 509
- hard resource usage limits, example 513
- hardware host ID, displaying (lmhostid) 153
- hardware host name 145
- health, hosts (see "hosts, health")
- hierarchical fairshare 306
- hierarchical share tree 308
- HIST_HOURS parameter in lsb.params, fairshare dynamic user priority 300
- historical run time 320
- history, job arrays 464, 466
- HJOB_LIMIT parameter in lsb.queueues 377
- hname static resource 240
- home directories
 - remote file access 657
 - shared 32
- \$HOME/.lsbatch directory 32
- \$HOME/.rhosts file
 - disadvantages 643
 - file transfer with lsrcp command 659
 - host authentication 643
- hopen badmin command 62
- Host, service instance property 194
- host affinity, same string 292
- host dispatch windows 420
- host entries, examples 80
- host exception handling
 - configuring 88
 - example 89
 - job exit rate exception 88, 124
- host failure 34, 694
- host groups 57
 - CONDENSE keyword 85
 - condensed, viewing 57
 - configuring external host groups 86
 - defining 131
 - defining condensed 85
 - external 86
 - overview 131
- host ID 145
 - displaying (lmhostid) 153
- host identifier 145
- host load levels 31
- host model static resource 240
- host models
 - adding custom names in lsf.shared 75
 - DEFAULT 678
 - select string 284

Index

- tuning CPU factors 87
- host name static resource 240
- host names
 - /etc/hosts file 77
 - aliases 77
 - matching with Internet addresses 77
 - ranges as aliases 78
 - resolv.conf file 79
 - resolver function 79
 - using DNS 79
 - wildcards and special characters 84
- host partition fairshare 301
- host redirection 711
- host reservation. *See* advance reservation
- host selection 283
- host state. *See* host status
- host status
 - busy
 - load index 237
 - lload 55
 - closed 57
 - bhosts 54
 - description 53
 - index 237
 - lockU and lockW 55, 237
 - ok 55, 237
 - ok
 - bhosts 53
 - load index 237
 - lload 55
 - unavail 53
 - load index 237
 - lload 55
 - unlicensed 54, 55, 237
 - unreach 54
- host thresholds 31
- host type static resource 240
- host types
 - adding custom names in lsf.shared 74
 - DEFAULT 678
 - resource requirements 279
 - select string 284
- host-based resources, description 232
- host-level
 - fairshare scheduling 301
 - resource information 399
- host-locked software licenses 261
- HOSTRESORDER environment variable 675
- hosts
 - adding with lsfinstall 64
 - associating resources with 251
 - closing 61
 - connecting to remote 714
 - controlling 61
 - copying files across 577
 - dispatch windows 62
 - displaying 57
 - file 79
 - finding resource 578
 - for advance reservations 407
 - health 187
 - logging on the least loaded 578
 - master candidates 631
 - multiple network interfaces 79
 - official name 77
 - opening 61
 - preselecting masters for LIM 630
 - redirecting 711
 - removing 66
 - resource groups overview 172
 - restricting use by queues 99
 - selecting for task 574
 - setting up 64, 65
 - spanning with parallel jobs 474
 - specifying master candidates 631
 - specifying master host 665
 - troubleshoot 187
 - viewing
 - architecture information 58
 - detailed information 57
 - eligible 31
 - exceptions 60
 - history 59
 - job exit rate and load 60
 - load by host 58, 235
 - load by resource 231
 - model and type information 59
 - resource allocation limits (blimits) 384
 - shared resources 234
 - status of closed servers 57
 - suspending conditions 526
- hosts file (/etc/hosts)
 - example host entries 80
 - host naming 77
 - name lookup 79
 - troubleshooting 674
- hosts file (LSF), configuring 80
- HOSTS parameter
 - in lsb.hosts 83
 - in lsb.queues file 83
- hosts.equiv file
 - host authentication 642
 - using rcp 659
- hostsetup script 64, 65
- hrestart badadmin command 46
- hshutdown badadmin command 46
- hstartup badadmin command 46

- I**
- %l substitution string in job arrays 462
- icon height, hosts 187
- idle job exceptions
 - configuring 100
 - description 100, 124
 - viewing with bjobs 108
 - viewing with bqueues 94
- idle time
 - built-in load index 238
 - description 565
 - suspending conditions 525
- if-else constructs
 - creating 273
 - files 271
- INCREMENT lines 149
- index list for job arrays 460
- input and output files
 - and interactive jobs 563
 - job arrays 461
 - splitting stdout and stderr 564
 - spooling directory 653
- Inst/Slot 193
- installation, on non-shared file systems 656
- installation (demo) 142
- installation (permanent) 144
- installation directories
 - default UNIX structure 43
 - Windows default structure 44
- interactive jobs
 - competing for software licenses 265
 - configuring queues to accept 562
 - redirecting scripts to standard input 569
 - resource reservation 291
 - running X applications 568
 - scheduling policies 561
 - specifying embedded submission options 570
 - specifying job options in a file 569
 - specifying shell 570
 - splitting stdout and stderr 564
 - spooling job command files 569
 - submitting 562
 - submitting and redirecting streams to files 563
 - submitting with pseudo-terminals 563
 - viewing queues for 562
 - writing job file one line at a time 569
 - writing job scripts 568
- interactive sessions, starting 578
- interactive tasks, file access 577
- interfaces, network 79
- Internet addresses, matching with host names 77
- Internet Domain Name Service (DNS), host naming 77
- inter-queue priority 523
- interruptible backfill 488
- io load index 239
- IPv6
 - configure hosts 82
 - supported platforms 82
 - using IPv6 addresses 82
- IRIX
 - Comprehensive System Accounting (CSA), configuring 662
 - utmp file registration 571
- it load index
 - automatic job suspension 524
 - description 238, 565
 - suspending conditions 525
- J**
- %J substitution string in job arrays 462
- JL/P parameter in lsb.users 377
- JL/U parameter in lsb.hosts 377
- job array dependency conditions, operators 463
- job arrays
 - %l substitution string 462
 - %J substitution string 462
 - argument passing 462
 - controlling 466
 - creating 459
 - dependency condition operators 463
 - dependency conditions 463
 - file preparation 461
 - format 460
 - history 464, 466
 - index list 460
 - input and output files 461
 - maximum size 461
 - monitoring 464, 466
 - overview 459
 - passing arguments 462
 - redirection of input and output 461
 - requeueing 466
 - specifying job slot limit 467
 - standard input and output 462
 - status 464, 466
 - submitting 459
 - syntax 460
- job checkpoint and restart
 - checkpoint directory 448
 - echkpt 449
 - erestart 449
 - job restart description 449
 - queue level 448
- job chunking. *See* chunk jobs
- job control actions
 - CHKPNT 556
 - configuring 555

Index

- default actions 553
- LS_EXEC_T 553
- on Windows 555
- overriding terminate interval 554
- RESUME 554
- SUSPEND 553
- TERMINATE 554
- terminating 557
- using commands in 556
- with lscsh 712
- job data transformer 606, 607
 - schedule 614
- job dependencies, logical operators 424
- job dependency conditions
 - advanced 426
 - description 424
 - done 425
 - ended 425
 - examples 426
 - exit 425
 - external 425
 - job arrays 463
 - job name 426
 - post_done 426, 531
 - post_err 426, 531
 - post-processing 531
 - scheduling 423
 - specifying 423
 - specifying job ID 426
 - started 426
- job dispatch
 - fast 585
 - maximum per session 585
- job dispatch order, fairshare 300
- job email
 - bsub options 649
 - disabling batch job notification 650
 - limiting size with LSB_MAILSIZE_LIMIT 650
- job exception handling
 - configuring 100
 - default eadmin action 124
 - exception types 100, 123
 - viewing exception status with bjobs 108
 - viewing exceptions with bqueues 94
- job execution environment 32
- job exit rate exceptions
 - configuring 88
 - description 88, 124
 - viewing with bhosts 60
- job file spooling
 - See also command file spooling
 - default directory 653
 - description 652
 - JOB_SPOOL_DIR parameter in lsb.params 652
- job files 29
- job groups
 - add limits 122
 - controlling jobs 120
 - default job group 115
 - description 114
 - displaying SLA service classes 340
 - example hierarchy 117
 - job limits 116
 - modify limits 123
 - viewing 119
- job idle factor, viewing with bjobs 108
- job ladders. See batch jobs, pre-execution commands
- job limit 519
- job limits, job groups 116
- job migration
 - absolute job priority scheduling 440
 - configuration
 - host level 450
 - queue level 450
 - description 450
 - manual 450
- job overrun exceptions
 - configuring 100
 - description 100, 124
 - viewing with bjobs 108
 - viewing with bqueues 94
- job preemption
 - absolute job priority scheduling 440
 - description 277
 - SLA scheduling 339
- job requeue
 - absolute job priority scheduling 439
 - automatic 441
 - exclusive 442
 - reverse requeue 442
 - user-specified 443
- job rerun
 - absolute job priority scheduling 439
 - disabling post-execution commands 445, 532
- job scripts, writing for interactive jobs 568
- job slot limits
 - for job arrays 467
 - for parallel jobs 472
 - viewing resource allocation limits (blimits) 384
- job spanning 283, 291
- job starters
 - augmentstarter 541
 - command-level 537
 - lic_starter script to manage software
 - licenses 265
 - preservestarter 541
 - queue-level
 - configuring 540

- description 538
 - specifying command or script 539, 540
 - user commands 540
- job states
 - description 104
 - DONE
 - description 103
 - post-execution commands 531
 - EXIT
 - abnormal job termination 105, 694
 - pre- and post-execution commands 531
 - PEND 103
 - POST_DONE 106, 531
 - POST_ERR 106, 531
 - post-execution 531
 - PSUSP 104
 - RUN 103
 - SSUSP 104
 - USUSP 104
 - WAIT for chunk jobs 456
- job submission 27
- JOB_ACCEPT_INTERVAL parameter in `lsb.params` 30
- JOB_CONTROLS parameter in `lsb.queues` 555
- JOB_EXIT_RATE_DURATION parameter in `lsb.params` 88
- JOB_IDLE parameter in `lsb.queues` 100
- JOB_INCLUDE_POSTPROC parameter, in `lsb.applications` 534
- JOB_OVERRUN parameter in `lsb.queues` 100
- JOB_POSITION_CONTROL_BY_ADMIN parameter in `lsb.params` 588
- JOB_POSTPROC_TIMEOUT parameter, in `lsb.applications` 535
- JOB_SCHEDULING_INTERVAL parameter in `lsb.params` 30, 587
- JOB_SPOOL_DIR parameter in `lsb.params` 652
- JOB_STARTER, `lsb.queues` file 540
- JOB_STARTER parameter in `lsb.queues` 540
- JOB_TERMINATE_INTERVAL parameter in `lsb.params` 517, 554
- JOB_UNDERRUN parameter in `lsb.queues` 100
- job-level
 - post-execution commands, description 530
 - pre-execution commands
 - configuring 531
 - description 530
 - resource requirements 281
 - resource reservation 388
 - run limits 518
- job-level suspending conditions, viewing 526
- jobs
 - changing execution order 109
 - checkpointing, chunk jobs 457
 - CHKPNT 556
 - controlling, in an application profile 359
 - dispatch order 30
 - email notification
 - disabling 650
 - options 649
 - enabling rerun 444
 - enforcing memory usage limits 517
 - exit codes 668
 - forcing execution 111
 - interactive. *See* interactive jobs
 - killing 112
 - in an application profile 359
 - limiting processors for parallel 478
 - modifying, in an application profile 359
 - optimum number running in SLA 328
 - pending 104
 - preemption 523
 - preemptive and preemptable 277
 - requeueing 466
 - requeueing, description 443
 - rerunning 444
 - rerunning automatically 444
 - restarting, automatically 444
 - resuming 111, 526, 527
 - in an application profile 359
 - sending specific signals to 113
 - short running 453
 - specifying options for interactive 569
 - specifying shell for interactive 570
 - spooling command and job files 569
 - spooling input, output, and command files 652
 - stopping, in an application profile 359
 - submitting
 - to a job group 118
 - to a service class 329
 - to an application profile 359
 - suspended 526
 - suspending 111, 523
 - suspending at queue level 526
 - switching queues 110
 - viewing
 - by user 106
 - configuration parameters in `lsb.params` 39
 - order in queue 31
 - viewing resource allocation limits (`blimits`) 384
- jobs command in `lscsh` 712
- jobs requeue, description 441
- JPRIORITY absolute job priority scheduling factor 434
- JSDL
 - configuration 252
 - elim for 258
 - required resources 251
- JSDL (Job Submission Description Language)
 - benefits 495

Index

- elim.jsdl 504
 - how to submit a job 504
 - LSF extension elements 501
 - schema files 495
 - supported elements 495
 - unsupported elements 503
 - using with LSF 495
- L**
- lic_starter script, to manage software licenses 265
 - license file (demo) 138
 - license file (permanent) 139
 - license host 140
 - license key, calculating checksum (lmcksum) 153
 - license management commands 151
 - license server
 - calculating license key check sums (lmcksum) 153
 - checking status (lmstat) 151
 - displaying
 - hardware host ID (lmhostid) 153
 - status (lmstat) 151
 - version information (lmver) 153
 - specifying TCP port 168
 - utility commands 151
 - license server daemon (lmgrd) 140
 - license server host 148
 - description 140
 - license.dat (demo) 138
 - license.dat (permanent) 139
 - license.dat file, location 146
 - license.dat, updating 149
 - licenses
 - advance reservation 405
 - cluster reconfiguration 52
 - displaying
 - FLEXlm version information (lmver) 153
 - hardware host ID (lmhostid) 153
 - LSF products 156
 - server status (lmstat) 153
 - floating LSF client
 - description 160
 - enabling 161
 - specifying host or model type 161
 - removing feature (lmremove) 153
 - rereading license file (lmreread) 153
 - shutting down FLEXlm server (lmdown) 153
 - software
 - counted 261
 - dedicated queue for 264
 - floating 262
 - host locked 261
 - interactive jobs and 265
 - LIM (Load Information Manager)
 - configuring number of restarts 258
 - configuring wait time 258
 - logging load information 258
 - preselecting master hosts 630
 - tuning 630
 - load indices 629
 - load thresholds 629
 - policies 628
 - run windows 628
 - LIM, master 140
 - limdebug command 687
 - limitations
 - lsrccp command 658
 - on chunk job queues 455
 - limits
 - job group 116
 - See resource allocation limits or resource usage limits
 - limtime command 690
 - Linux and Windows hosts, multicore processor support 137
 - Linux hosts, dual-core processor support 136
 - lmcksum command 153
 - lmdown command 153
 - lmgrd daemon 140
 - lmhostid 145
 - lmhostid command 153
 - lmremove command 153
 - lmreread command 153
 - lmstat command
 - description 153
 - example 152
 - using 151
 - lmver command 153
 - load average 238
 - load indices
 - See also resources
 - built-in
 - overriding 258
 - summary 237
 - io 239
 - it 238
 - ls 238
 - mem 239
 - pg 238
 - r15m 238
 - r15s 238
 - r1m 238
 - swp 239
 - tmp 239
 - tuning for LIM 629
 - types 565
 - ut 238
 - viewing 38, 239

- load levels
 - viewing by resource 231
 - viewing for cluster 38
 - viewing for hosts 58
- load sharing
 - displaying current setting 713
 - with lscsh 714
- load thresholds
 - configuring 525
 - description 280
 - paging rate, tuning 630
 - queue level 525
 - tuning 629
 - tuning for LIM 628, 629
- loader controller 604, 606, 607
 - data collection
 - disabling 616
 - frequency 608, 616
- local event logging, mbatchd restart with MAX_INFO_DIRS 590
- local mode in lscsh 708
- locality, parallel jobs 291, 474
- location of license (demo) 143
- location of license (permanent) 146
- lockU and lockW host status
 - lslod command 55
 - status load index 237
- log file (FLEXlm) 152
- log files
 - about 202
 - default location 201, 661
 - directory permissions and ownership 202, 662
 - ESC 202
 - logging events on Cray UNICOS 662
 - maintaining 201, 662
 - managing 201, 662
 - mbatchd.log.host_name 201, 662
 - mbschd.log.host_name 201, 662
 - named 202
 - PEM 202
 - res.log.host_name 201, 662
 - retrieve 206
 - sbatchd.log.host_name 201, 662
 - troubleshooting 206
 - VEMKD 202
 - WSG 202
 - WSM 202
- log on, Management Console
- LOG_DAEMON facility, LSF error logging 201, 663
- logging classes, description 203
- logging levels, description 203
- logical operators
 - in time expressions 270
 - job dependencies 424
- login sessions 238
- login shell, using lscsh as 710
- logs
 - classes 203
 - entry formats 203
 - levels 203
- lost_and_found queue 98
- ls load index 238
- ls_connect API call 544
- LS_EXEC_T environment variable 553
- LS_JOBPID environment variable 552
- ls_postevent() arguments 624
- lsadmin command
 - limlock 54
 - limunlock 54
- lsb.applications file
 - adding an application profile 358
 - JOB_INCLUDE_POSTPROC parameter 534
 - JOB_POSTPROC_TIMEOUT parameter 535
 - NAME parameter 358
 - POST_EXEC parameter 532
 - PRE_EXEC parameter 532
- lsb.events file
 - event logging 33
 - logging administrator comments
 - for host open and close 62
 - for mbatchd restart 48
 - for queue events 95
 - managing event log 663
- lsb.hosts file
 - CONDENSE keyword 85
 - host exception handling 88
 - if-else constructs 271
 - time-based configuration 271
 - user groups 132
 - USER_SHARES parameter 132
 - using host groups 83
 - using user groups 132
- lsb.modules file
 - advance reservation 405
 - schmod_advrsv plugin 405
- lsb.params file
 - absolute run time limit 519
 - CHUNK_JOB_DURATION parameter 454
 - CLEAN_PERIOD parameter 113
 - CONDENSE_PENDING_REASONS parameter 588
 - controlling lsb.events file rewrites 663
 - CPU time normalization 516
 - default application profile 358
 - default normalization host 516
 - default SLA 344
 - DEFAULT_JOBGROUP parameter 115
 - DEFAULT_SLA_VELOCITY parameter 351
 - EADMIN_TRIGGER_DURATION threshold for ex-

Index

- ception handling 101
- ENABLE_EXIT_RATE_PER_SLOT parameter 127
- GLOBAL_EXIT_RATE parameter 127
- if-else constructs 271
- job termination signal interval 517
- JOB_ACCEPT_INTERVAL parameter 30
- JOB_EXIT_RATE_DURATION for exception handling 88
- JOB_POSITION_CONTROL_BY_ADMIN parameter 588
- JOB_SCHEDULING_INTERVAL parameter 30, 587
- JOB_SPOOL_DIR parameter 652
- MAX_CONCURRENT_JOB_QUERY parameter 587
- MAX_INFO_DIRS parameter 589
- MAX_PEND_JOBS parameter 104
- MAX_SBD_CONNS parameter 586
- MBD_EGO_CONNECT_TIMEOUT parameter 351
- MBD_EGO_READ_TIMEOUT parameter 351
- MBD_EGO_TIME2LIVE parameter 351
- MBD_QUERY_CPUS parameter 638
- MBD_REFRESH_TIME parameter 635
- MBD_USE_EGO_MXJ parameter 352
- MIN_REFRESH_TIME parameter 636
- MIN_SWITCH_PERIOD parameter 588
- NEWJOB_REFRESH parameter 638
- non-normalized run time limit 519
- PARALLEL_SCHED_BY_SLOT parameter 472
- SLA_TIMER parameter 352
- specifying job input files 652
- SUB_TRY_INTERVAL parameter 104
- time-based configuration 271
- lsb.queues file
 - adding a queue 98
 - HOSTS parameter 83
 - if-else constructs 271
 - job exception handling 100
 - JOB_IDLE parameter 100
 - JOB_OVERRUN parameter 100
 - JOB_UNDERRUN parameter 100
 - normalization host 521
 - POST_EXEC parameter 531
 - PRE_EXEC parameter 531
 - QUEUE_NAME parameter 98
 - resource usage limits 513
 - restricting host use by queues 99
 - time-based configuration 271
 - user groups 132
 - USERS parameter 132
 - using host groups 83
 - using user groups 132
- lsb.queues files, DEFAULT_HOST_SPEC parameter 521
- lsb.resources file
 - advance reservation policies 405
 - if-else constructs 271
 - parameters 375
 - time-based configuration 271
 - viewing limit configuration (blimits) 384
- lsb.serviceclasses file
 - configuring SLA scheduling 330
 - CONSUMER parameter 345
 - CONTROL_ACTION 339
 - EGO_RES_REQ parameter 351
 - MAX_HOST_IDLE_TIME parameter 351
- lsb.users file
 - if-else constructs 271
 - MAX_PEND_JOBS parameter 104
 - time-based configuration 271
 - user groups 132
 - USER_NAME parameter 132
 - using user groups 132
- LSB_CHUNK_RUSAGE parameter in lsf.conf 511
- LSB_CONFDIR parameter in lsf.conf, default UNIX directory 43
- LSB_DEFAULT_JOBGROUP environment variable 115
- LSB_DEFAULTQUEUE environment variable 28
- LSB_DISABLE_RERUN_POST_EXEC parameter in lsf.conf 445, 532
- LSB_ESUB_METHOD in lsf.conf 550
- LSB_HOSTS environment variable 469
- LSB_JOB_CPULIMIT parameter in lsf.conf 516
- LSB_JOBEXIT_STAT environment variable 532
- LSB_JOBINDEX environment variable 462
- LSB_JOBPEND environment variable 532
- LSB_JOBPGIDS environment variable 556
- LSB_JOBPIIDS environment variable 556
- LSB_LOCALDIR parameter in lsf.conf file 665
- LSB_MAILSIZE environment variable 650
- LSB_MAILSIZE_LIMIT parameter in lsf.conf 650
- LSB_MAILTO parameter in lsf.conf 649
- LSB_MAX_JOB_DISPATCH_PER_SESSION parameter in lsf.conf 585
- LSB_MBD_PORT parameter in lsf.conf 75
- LSB_NTRIES environment variable 104
- LSB_PRE_POST_EXEC_USER parameter, in lsf.sudoers 534
- LSB_QUERY_PORT parameter in lsf.conf 587, 637
- LSB_REQUEUE_TO_BOTTOM parameter in lsf.conf 441, 442
- LSB_SBD_PORT parameter in lsf.conf 75
- LSB_SHARED_DIR parameter in lsf.conf
 - default UNIX directory 43
 - duplicate event logging 664
- LSB_SHARED_DIR/cluster_name/logdir, LSF log files 201, 661

- LSB_SIGSTOP parameter in `lsf.conf` 111, 557
- LSB_SUB_ABORT_VALUE environment variable 547
- LSB_SUB_COMMANDNAME parameter in `lsf.conf` 545
- LSB_SUB_PARM_FILE environment variable 544
- LSB_SUSP_REASON environment variable 556
- LSB_SUSP_SUBREASONS environment variable 556
- LSB_UTMP parameter in `lsf.conf` 571
- `.lsbatch` directory 32
- LSF Daemon Error Log 201, 662
- LSF events
 - generated by LSF 624
 - generation of 623
 - program arguments 624
- LSF features 147
- LSF license grace period 141
- LSF licenses, license file location 146
- LSF licensing
 - banded license types 137
 - dual-core processor Linux hosts 136
 - `lsf_dualcore_x86` feature 136
 - `lsf_mv_grid_filter` feature 137
 - multicore processor Linux and Windows hosts 137
- LSF master LIM 140
- LSF parameters. *See* individual parameter names
- LSF products 147
 - displaying enabled license 156
- LSF vendor license daemon (`lsf_ld`) 140
- LSF version, viewing 38
- `lsf.cluster.cluster_name` file
 - ADMINISTRATORS parameter 45
 - configuring cluster administrators 45
 - exclusive resources 286
 - floating LSF client licenses 161
 - license checkout 141
- `lsf.conf` file
 - comprehensive system account 662
 - configuring duplicate logging 665
 - configuring TCP service ports 75
 - custom file transfer 659
 - daemon service ports 75
 - default UNIX directory 43
 - duplicate event logging 664
 - dynamic host startup time 67
 - limiting the size of job email 650
 - LSB_CHUNK_RUSAGE parameter 511
 - LSB_DISABLE_RERUN_POST_EXEC parameter 445, 532
 - LSB_JOB_CPULIMIT parameter 516
 - LSB_JOB_MEMLIMIT 517
 - LSB_MAILSIZE_LIMIT parameter 650
 - LSB_MAILTO parameter 649
 - LSB_MAX_JOB_DISPATCH_PER_SESSION parameter 585
 - LSB_MEMLIMIT_ENFORCE 517
 - LSB_QUERY_PORT parameter 587, 637
 - LSB_SIGSTOP parameter 111
 - LSB_SUB_COMMANDNAME parameter 545
 - LSF_BINDIR parameter 43, 659
 - LSF_DYNAMIC_HOST_WAIT_TIME parameter 67
 - LSF_ENABLE_CSA parameter 662
 - LSF_LOG_MASK parameter 663
 - LSF_LOGDIR parameter 201, 663
 - LSF_MANDIR parameter 43
 - LSF_MASTER_LIST parameter 67, 631
 - LSF_MISC parameter 43
 - LSF_SERVERDIR parameter 43
 - LSF_STRICT_CHECKING parameter 71
 - `lsrcp` command executable 659
 - managing error logs 663
 - master host candidates 631
 - per-job CPU limit 516
 - resource usage limits for chunk jobs 511
 - sending email to job submitter 649
 - setting message log to debug level 687
 - strict checking, enabling 71
- `lsf.conf` parameter LSF_LICENSE_FILE 146
- `lsf.licensescheduler` file
 - if-else constructs 271
 - time-based configuration 271
- `lsf.shared` file
 - adding a custom host type and model 74
 - tuning CPU factors 86
- `lsf.sudoers` file, LSB_PRE_POST_EXEC_USER parameter 534
- LSF_BINDIR parameter in `lsf.conf` 43, 659
- LSF_CONFDIR parameter in `lsf.conf` 43
- `lsf_dualcore_x86` license feature 136
- LSF_DYNAMIC_HOST_WAIT_TIME parameter in `lsf.conf` 67
- LSF_ENABLE_CSA parameter in `lsf.conf` 662
- LSF_INCLUDEDIR parameter in `lsf.conf`, default UNIX directory 43
- LSF_JOB_STARTER environment variable 539
- `lsf_ld` 140
- LSF_LICENSE_FILE parameter in `lsf.conf` 146
- LSF_LIM_PORT parameter in `lsf.conf` 75
- LSF_LOG_MASK parameter in `lsf.conf` 663, 687
- LSF_LOGDIR parameter in `lsf.conf` 201, 663
- LSF_MANDIR parameter in `lsf.conf` 43
- LSF_MASTER_LIST parameter in `lsf.conf` 67, 631
- LSF_MISC parameter in `lsf.conf` 43
- `lsf_mv_grid_filter` license feature 137
- LSF_NT2UNIX_CLTRB environment variable 558
- LSF_NT2UNIX_CLTRC environment variable 558
- LSF_RES_PORT parameter in `lsf.conf` 75
- LSF_RSH parameter in `lsf.conf`, controlling

Index

- daemons 46
- LSF_Sched_Advance_Reservation license 405
- LSF_SERVERDIR 148
- LSF_SERVERDIR parameter in `lsf.conf` 43
- LSF_STRICT_CHECKING parameter in `lsf.conf` 71
- LSF_TOP directory, default UNIX directory structure 43
- lsfbhostsloader 605
- lsfeventsloader 605
- lsfinstall, adding a host 64
- lsfinstall program 142
- lsfshutdown command, shutting down daemons on all hosts 46
- lsfslaloader 605
- lsfstartup command, starting daemons on all hosts 46
- lshosts, viewing dynamic host information 61
- lshosts command, DEFAULT host model or type 678
- lshosts -l, viewing licensed LSF products 156
- lsrnp command
 - description 657
 - executable file location 659
 - file transfer 658
 - restrictions 658
- lstcsh
 - about 707
 - difference from other shells 709
 - exiting 710
 - limitations 709
 - local mode 708
 - remote mode 708
 - resource requirements 708
 - starting 710
 - task lists 708
 - using as login shell 710
 - writing shell scripts in 714
- M**
- mail
 - disabling batch job notification 650
 - job options 649
 - limiting the size of job email 650
- Management Console, logging on 184
- ManagementServices, about 189
- mandatory esub method (LSB_ESUB_METHOD) 550
- mandatory first execution host, parallel jobs 473
- master candidates, setting 197
- master esub (mesub)
 - configuring 551
 - description 550
- master host candidates 196
 - with LSF_MASTER_LIST 631
- master host failover, about 196
- master hosts 196
 - displaying properties 187
 - in non-shared file systems 656
 - preselecting 630
 - specifying 665
 - viewing current 38
- Max Inst 192
- MAX_CONCURRENT_JOB_QUERY parameter in `lsb.params` 587
- MAX_HOST_IDLE_TIME parameter in `lsb.serviceclasses` 351
- MAX_INFO_DIRS parameter in `lsb.params` 589
- MAX_JOB_NUM parameter in `lsb.params` 663
- MAX_JOBS parameter in `lsb.users` 377
- MAX_PEND_JOBS parameter in `lsb.params` or `lsb.users` 104
- MAX_SBD_CONNS parameter in `lsb.params` 586
- maximum
 - number of processors for parallel jobs 472
 - resource usage limit 513
 - run limit 513
- maxmem static resource 240
- maxswp static resource 240
- maxtmp static resource 240
- mbatchd (master batch daemon)
 - expiry time 636
 - push new job information to a child
 - mbatchd 636, 638
 - refresh time 636
 - restarting 48
 - shutting down 49
 - specifying query-dedicated port 636
 - specifying time interval for forking child 636
 - tuning on UNIX 635
- mbatchd.log.host_name file 201, 662
- MBD. *See* mbatchd
- MBD_EGO_CONNECT_TIMEOUT parameter in `lsb.params` 351
- MBD_EGO_READ_TIMEOUT parameter in `lsb.params` 351
- MBD_EGO_TIME2LIVE parameter in `lsb.params` 351
- MBD_QUERY_CPUS parameter in `lsb.params` 638
- MBD_REFRESH_TIME parameter in `lsb.params` 635
- MBD_USE_EGO_MXJ parameter in `lsb.params` 352
- mbddebug command 687
- mbdrestart badmin command 46
- mbdtime command 690
- mbschd.log.host_name file 201, 662
- MEM absolute job priority scheduling factor 434
- mem load index, description 239
- MEMLIMIT parameter in `lsb.queues` 517
- memory
 - available 239
 - usage limit 517
 - viewing resource allocation limits (blimits) 384

- mesub (master esub)
 - configuring 551
 - description 550
- migrated jobs, absolute job priority scheduling 440
- Min Inst 192
- MIN_REFRESH_TIME parameter in lsb.params 636
- MIN_SWITCH_PERIOD parameter in lsb.params 588
- minimum processors for parallel jobs 472
- missed SLA scheduling goals, control action 339
- model static resource 240
- multicore processor support 137
- multi-homed hosts 79
- multiple condensed host groups 86
- multiple conditions, dependency expressions 424
- multiple esub 550
- multiple license server hosts 154
- multiple queues, absolute job priority scheduling 436
- multi-processor hosts 141
- multiprocessor hosts
 - configuring queue-level load thresholds 525
 - tuning LIM 630
- multithreading, configuring mbatchd for 635
- MXJ parameter in lsb.hosts 377
- N**
- Name, service instance property 194
- name lookup using /etc/hosts file 79
- NAME parameter in lsb.applications 358
- native language system, and lscsh 709
- n_cpus static resource
 - dynamically changing processors 246
 - reported by LIM 240
- ndisks static resource 240
- network
 - failure 33
 - interfaces 79
 - partitioning, and duplicate event logging 664
 - port numbers, configuring for NIS or NIS+ databases 76
- Network File System. *See* NFS
- Network Information Service. *See* NIS
- NEWJOB_REFRESH parameter in lsb.params 638
- NFS (Network File System)
 - automount command 655, 672
 - nosuid option 643
 - overview 655
- NIS (Network Information Service)
 - configuring port numbers 76
 - host name lookup in LSF 77
 - ypcat hosts.byname 79
- non-normalized run time limit 519
- non-shared file systems, installing LSF 656
- normalization
 - CPU time limit 521
 - host 521
 - run time limit 521
- normalization host 516
- normalized run queue length
 - description 238
 - tuning LIM 630
- nosuid option, NFS mounting 643
- NOT operator (!), job dependencies 424
- not operator (~)
 - host partition fairshare 302
 - host-based resources 252
- NQS (Network Queueing System), logging events on Cray UNICOS 662
- nqsacct file 662
- NUM_RECALLED_HOSTS, bsla 347
- number of processors for parallel jobs 472
- numdone dependency condition 463
- numended dependency condition 463
- numerical resources 232
- numexit dependency condition 463
- numhold dependency condition 463
- numpend dependency condition 463
- numrun dependency condition 463
- numstart dependency condition 463
- O**
- obsolete parameters
 - FAIRSHARE_QUEUES 436
 - USER_ADVANCE_RESERVATION in lsb.params 406
- official host name 77
- ok host status
 - lsload command 55
 - status load index 237
- ok host status
 - bhosts command 53
 - lsload command 55
 - status load index 237
- one-time advance reservation 408
- operators
 - job array dependency conditions 463
 - logical in job dependencies 424
 - logical in time expressions 270
 - not (~)
 - host partition fairshare 302
 - host-based resources 252
 - relational, exit dependency condition 425
 - remote file access 657
 - resource requirements 285
 - selection strings 285
- OR operator (||), job dependencies 424
- order of job execution, changing 109
- order string 287

Index

- OS memory limit 517
- output and input files, for job arrays 462
- output file spooling, default directory 653
- overrun job exceptions
 - configuring 100
 - description 100, 124
 - viewing with bjobs 108
 - viewing with bqueues 94
- ownership of log directory 202, 662
- P**
- paging rate
 - automatic job suspension 524
 - checking 525
 - description 238, 565
 - load index 238
 - suspending conditions 525
- parallel fairshare 492
- parallel jobs
 - allocating processors 470
 - backfill scheduling 483
 - deadline constraint scheduling 493
 - fairshare 492
 - interruptible backfill scheduling 488
 - job slot limits 472
 - limiting processors 478
 - locality 291, 474
 - mandatory first execution host 473
 - number of processors 472
 - overview 469
 - processor reservation 481
 - selecting hosts with same string 292
 - spanning hosts 474
 - submitting 470
- parallel programming, packages 470
- parallel tasks
 - running with lsgun 575
 - starting 471
- PARALLEL_SCHED_BY_SLOT parameter in lsb.params 472
- partial licensing 156
- partitioned networks 34
- password, change 185
- PATH environment variable
 - and lscsh 710
 - shared user directories 32
- paths
 - /etc/hosts file
 - example host entries 80
 - host naming 77
 - name lookup 79
 - /etc/hosts.equiv file 643
 - host authentication 642
 - using rcp 659
 - /etc/services file, adding LSF entries to 76
 - /net 656
 - /usr/bin/ 32
- PEND, job state 103
- pending jobs
 - absolute job priority scheduling 432
 - order of absolute job priority scheduling 437
- pending reasons
 - queue-level resource reservation 388
 - viewing 105
- per-CPU licensing 141
- PERF 604
- performance tuning
 - busy thresholds 628
 - LIM policies 628
 - load indices 629
 - load thresholds 629
 - mbatchd on UNIX 635
 - preselecting master hosts 630
 - run windows for LIM 628
- periodic tasks 201, 662
- per-job CPU limit 516
- permanent license 136
- permanent LSF license, displaying server status (lmstat) 151
- permissions, log directory 202, 662
- per-process limits
 - CPU limit 516
 - data segment size 516
 - file size 517
 - memory limit 517
 - stack segment size 519
- per-resource reservation
 - configuring 389
 - viewing with bresources 401
- pg load index, suspending conditions 525
- PIM (Process Information Manager), resource use 235
- PJOB_LIMIT parameter in lsb.queues 377
- plclient 608, 610, 611
- PluginModule section in lsb.modules, advance reservation 405
- policies
 - fairshare 296
 - tuning for LIM 628
- port, notation, license daemon 168
- port (in LSF_LICENSE_FILE) 146
- port numbers, configuring for NIS or NIS+ databases 76
- ports
 - registering daemon services 75
 - specifying dedicated 636
- post_done job dependency condition 426, 531
- POST_DONE post-execution job state 106, 531
- post_err job dependency condition 426, 531

- POST_ERR post-execution job state 106, 531
- POST_EXEC parameter
 - in lsb.applications 532
 - in lsb.queues 531
- post-execution
 - job dependency conditions 531
 - job states 531
- post-execution commands
 - configuring 531
 - disabling for rerunnable jobs 445, 532
 - including in job processing 534
 - job-level 530
 - overview 529
 - queue-level 530
 - running under user ID 534
 - setting a post-processing timeout 535
- PRE_EXEC parameter
 - in lsb.applications 532
 - in lsb.queues 531
- PREEMPT_FOR parameter in lsb.params 494
- preemptable
 - jobs 278
 - queues 277
- preemption, absolute job priority scheduling 440
- preemption. *See* preemptive scheduling
- preemptive
 - jobs 277
 - queues 277
 - scheduling, description 277
- preemptive scheduling
 - advance reservation 415
 - SLA scheduling 339
- pre-execution commands
 - configuring 531
 - job-level 530
 - overview 529
 - queue-level 530
 - running under user ID 534
- preferences, setting for tables 187
- preservestarter job starter 541
- PRIORITY parameter in lsb.queues 305, 312
- priority user fairshare 325
- priority. *See* dynamic user priority
- PROC absolute job priority scheduling factor 434
- process allocation for parallel jobs 283, 292
- PROCESLIMIT parameter in lsb.queues 518
- processor reservation, configuring 482
- processors
 - limiting for parallel jobs 478
 - number for parallel jobs 472
 - reservation 481
- products 147
- PRODUCTS line, editing 147
- programs, handling LSF events 624
- project names, viewing resource allocation limits (blimits) 384
- pseudo-terminal
 - submitting interactive jobs with 563
 - using to run a task 575
- PSUSP job state
 - description 111
 - overview 104
- Q**
- qact badmin command 95
- qclose badmin command 95
- qinact badmin command 95
- QJOB_LIMIT parameter in lsb.queues 377
- qopen badmin command 95
- QPRIORITY absolute job priority scheduling factor 434
- queue dispatch windows 420
- queue groups, absolute job priority scheduling 436
- queue priority 28
- queue thresholds, viewing 31
- QUEUE_GROUP parameter in lsb.queues 436
- QUEUE_NAME parameter in lsb.queues 98
- queue-based fairshare, resource usage measurement 299
- queue-level
 - fairshare across queues 302
 - fairshare scheduling 302
 - job starter 540
 - pre- and post-execution commands
 - configuring 531
 - description 530
 - resource limits 513
 - resource requirements 280
 - resource reservation 388
 - run limits 514
- queue-level resource information, viewing 400
- queue-level resource limits, defaults 513
- queues
 - adding and removing 98
 - automatic selection 29
 - backfill queue 485
 - changing job order within 109
 - chunk job limitations 455
 - configuring
 - for chunk jobs 454
 - job control actions 555
 - suspending conditions 526
 - default 29
 - dispatch windows 96
 - fairshare across queues 302
 - interactive 562
 - interruptible backfill 489
 - lost_and_found 98

Index

- overview 27
 - preemptive and preemptable 277
 - QUEUE_EXIT_VALUES parameter 532
 - restricting host use 99
 - run windows 97
 - setting rerun level 444
 - specifying suspending conditions 526
 - viewing
 - available 92
 - default 28
 - detailed queue information 92
 - for interactive jobs 562
 - history 93
 - job exception status 94
 - resource allocation limits (blimits) 384
 - status 92
 - viewing absolute job priority scheduling information 438
 - viewing administrator of 38
- R**
- R res_req command argument 282
 - r15m load index
 - built-in resources 238
 - description 566
 - suspending conditions 525
 - r15s load index
 - built-in resources 238
 - description 566
 - suspending conditions 525
 - r1m load index
 - built-in resources 238
 - description 566
 - suspending conditions 525
 - ranges, host name aliases 78
 - rcp command 657
 - RECALLED_HOSTS_TIMEOUT, bsla 347
 - reclaim grace period, consumer properties 190
 - recurring advance reservation 409
 - redundant license server hosts 155
 - relational operators, exit dependency condition 425
 - remote execution, with lscsh 709
 - remote file access, operators 657
 - remote jobs
 - bringing background jobs to foreground 712
 - execution priority 240
 - remote mode in lscsh 708
 - removing
 - a licensed feature (lremove) 153
 - reports (reporting feature) 618
 - reports 597
 - architecture 604
 - creating 600, 601, 602
 - custom 598, 600
 - creating 600, 601, 602
 - deleting 600, 603
 - producing 602
 - data loader plug-ins 607
 - EGO 605
 - log files 608, 611
 - LSF 604
 - data purger 606, 608
 - record expiry time 615
 - schedule 608, 613
 - database 598, 607, 609
 - Derby 598, 609
 - moving 609, 619
 - MySQL 621
 - Oracle 224, 225, 620
 - schema 224, 620, 621
 - deleting 600, 603
 - disabling 618
 - event data files 609
 - EGO 612
 - LSF 612
 - exporting 599, 600, 601, 603
 - job data transformer 606, 607
 - schedule 614
 - loader controller 604, 606, 607
 - data collection 608, 616
 - log files 610
 - plclient 608, 610, 611
 - status 610
 - log files 607, 608, 611
 - data loader plug-ins 611
 - loader controller 610
 - PERF 604
 - producing
 - custom 602
 - standard 599
 - services 607
 - data purger 606, 608
 - database 607, 609
 - disabling automatic startup 610
 - job data transformer 606, 607
 - loader controller 606, 607
 - log files 607, 611
 - restarting 607, 609
 - standard 598
 - exporting 599, 601
 - producing 599
 - QUEUE_EXIT_VALUES parameter in lsb.queues 442
 - queued jobs
 - absolute job priority scheduling 439
 - automatic 441
 - description 441
 - exclusive 442

- reverse 442
 - user-specified 443
- rerun jobs, absolute job priority scheduling 439
- rerunnable jobs 444
 - chunk jobs 457
 - disabling post-execution 445, 532
- RERUNNABLE parameter in `lsb.queues` 444
- Res Req 193
- `res.log.host_name` file 201, 662
- `resdebug` command 687
- reservation, advance 404, 405
- reservation ID, advance reservation 413
- reserved memory, for pending jobs 400
- resetting LSF floating client 160
- `resolv.conf` file 79
- resolver function 79
- resource allocation limits
 - configuring 375
 - viewing (blimits) 384
- resource configurations, viewing with blimits 384
- resource distribution tree
 - about 172
 - about building 189
- resource granularity 389
- resource groups, about 172
- resource names
 - aliases 284
 - description 250
- resource plan, overview 172
- resource requirements
 - and task lists in `lscsh` 708
 - description 279
 - exclusive resources 286
 - for advance reservations 407
 - host type 279
 - operators 285
 - ordering hosts 283, 287
 - parallel job locality 283, 291
 - parallel job processes 283, 292
 - parallel jobs, selecting hosts 292
 - resource reservation 287
 - resource usage 283, 287
 - select string 284
 - selecting hosts 283, 284, 292
- resource reservation
 - absolute job priority scheduling 440
 - description 387
 - static shared resources 253
- resource usage
 - fairshare scheduling 298
 - resource requirements 283, 287
 - viewing 235
- resource usage limits
 - ceiling 513
- chunk job enforcement 511
 - configuring 513
 - conflicting 510
 - default 513
 - for deadline constraints 275
 - hard 513
 - maximum 513
 - priority 510
 - soft 513
 - specifying 513
- ResourceMap section in `lsf.cluster.cluster_name` 251
- ResourceReservation section in `lsb.resources` 405
- resources
 - See also* load indices
 - adding custom 250
 - advance reservations 404
 - associating with hosts 251
 - Boolean 232
 - built-in 237
 - configuring custom 250
 - configuring limits 375
 - custom 249
 - host-level 399
 - per-resource configuration 401
 - queue-level 400
 - shared 233, 234
 - types 232
 - viewing
 - available 38, 230
 - host load 231
 - shared 38
- Restarts, service property 193
- `restime` command 690
- restrictions
 - chunk job queues 455
 - `lsrnp` command 658
 - `lscsh` 709
- RESUME job control action 554
- resume thresholds, viewing 527
- RESUME_COND parameter in `lsb.queues` 554
- reverse requeue 442
- `rexpri` static resource 240
- RG 192
- `.rhosts` file
 - troubleshooting 674
 - disadvantages 643
 - file transfer with `lsrnp` 659
 - host authentication 643
- `rlogin` command, interactive terminals 565
- `rsh` command, `lsfrestart` 46
- RUN job state, overview 103
- run limits
 - ceiling 513
 - configuring 510, 518

Index

- default 514
- maximum 513
- specifying 520
- run queue
 - effective 238
 - normalized 238
 - suspending conditions 525
- run time
 - decayed 320
 - historical 320
 - normalization 521
- run time limit, non-normalized (absolute) 519
- run windows
 - description 419
 - queues 97
 - tuning for LIM 628
- RUN_JOB_FACTOR parameter in `lsb.params`, fairshare dynamic user priority 300
- RUN_TIME_FACTOR parameter in `lsb.params`, fairshare dynamic user priority 299
- RUN_WINDOW, queues 97
- RUNLIMIT parameter in `lsb.queues` 518
- running host name, service instance properties 194
- running jobs, viewing 104
- rusage
 - resource requirements section 283
 - resource reservation 387
 - usage string syntax 287
- S**
- same string 292
- sample `/etc/hosts` file entries 80
- SampleApplications, about 189
- sbatchd (slave batch daemon)
 - remote file access 656
 - restarting 47
 - shutting down 47
- sbatchd.log.host_name file 201, 662
- sbddebug command 687
- sbdtime command 690
- schddebug command 687
- schddtime command 690
- scheduling
 - exclusive 276
 - fairshare 296
 - hierarchical fairshare 306
 - preemptive, description 277
 - service level agreement (SLA) 30, 327
 - threshold
 - host selection 31
 - queue-level resource requirements 280
- scheduling policies, absolute job priority
- scheduling 431
- scheduling priority factors, absolute job priority
- scheduling 432
- schmod_adrvsv plugin for advance reservation 405
- S-Class LSF license type 137
- scripts
 - check_license for counted software licenses 262
 - lic_starter to manage software licenses 265
 - redirecting to standard input for interactive jobs 569
 - writing for interactive jobs 568
 - writing in `lscsh` 714
- SDK, defining demand 171
- security, LSF authentication 642
- selection strings
 - defined keyword 286
 - description 284
 - operators 285
- Sequence, service instance property 194
- server hosts, viewing detailed information 57
- SERVER line, `license.dat` file 139
- server static resource 240, 241
- server status closed 57
- servers, displaying license status (`lmstat`) 153
- service class
 - configuring 330
 - examples 331
 - goal-oriented scheduling 328
- service classes
 - bacct command 332, 334
 - bjgroup command 340
 - bjobs command 334
 - description 328
 - EGO-enabled SLA scheduling 341
 - submitting jobs 329
- service database examples 76
- service definitions, about 192
- service director, about 191
- service error messages 195
- service instance properties
 - Host 194
 - Name 194
 - Sequence 194
 - Start 194
 - State 194
- service instance sequence, service instance property 194
- service instances
 - about 193
 - definitions 194
 - states 193
- service level agreement. *See* SLA scheduling
- service level goals
 - job preemption 339
 - optimum number of running jobs 328
 - service classes 328

- service name, service instance property 194
- service ports (TCP and UDP), registering 75
- service properties
 - # of Inst 192
 - Consumer 192
 - Depend 193
 - Description 192
 - Inst/Slot 193
 - Max Inst 192
 - Min Inst 192
 - Res Req 193
 - Restarts 193
 - RG 192
 - state 192
 - Type 193
- services
 - about 172, 191
 - cluster
 - service director 172, 191
 - web service gateway 172, 191
 - WEBGUI 172, 191
 - states 191
- setuid permissions 674
- setup (demo) 142
- setup (permanent) 144
- SGI IRIX. *See* IRIX
- share assignments 297
- share tree 308
- shared file systems, using LSF without 656
- shared files 672
- shared resources
 - defined keyword 286
 - description 233
 - exclusive resourceselection strings, exclusive resources 286
 - static, reserving 253
 - viewing 234
- shared user directories 32
- sharedresproploader 605
- sharedresusageloader 605
- shares
 - fairshare assignment 297
 - viewing user share information 130
- shell mode, enabling 577
- shell scripts. *See* scripts
- shell variables and lscsh 709
- shells
 - default shell for interactive jobs 570
 - lscsh 709
 - specifying for interactive jobs 570
- short-running jobs, as chunk jobs 453
- shutting down, FLEXlm server (lmdown) 153
- SIGCONT signal
 - default RESUME action 554
- job control actions 113
- SIGINT signal
 - conversion to Windows 558
 - default TERMINATE action 554
 - job control actions 113
- SIGKILL signal
 - default TERMINATE action 554
 - job control actions 113
 - sending a signal to a job 113
- signals
 - avoiding job action deadlock 557
 - configuring SIGSTOP 111, 554, 557
 - converting 558
 - customizing conversion 558
 - job exit codes 668
 - sending to a job 113
 - SIGINT 113
 - SIGTERM 113
- SIGQUIT signal, conversion to Windows 558
- SIGSTOP signal
 - bstop 111
 - configuring 111, 554, 557
 - default SUSPEND action 554
 - job control actions 113
- SIGTERM signal
 - default TERMINATE action 554
 - job control actions 113
- SIGTSTP signal
 - bstop 111
 - default SUSPEND action 554
- site-defined resources, resource types 232
- SLA scheduling
 - bacct command 334
 - bjgroup command 340
 - bjobs command 334
 - bsla command 332
 - configuring 330
 - configuring EGO-enabled SLA service classes 344
 - deadline goals 328
 - default SLA for EGO-enabled SLA scheduling 344
 - delayed goals 339
 - description 327
 - EGO-enabled SLA scheduling 341
 - job preemption 339
 - missed goals 339
 - optimum number of running jobs 328
 - service classes
 - description 328
 - examples 331
 - service level goals 328
 - submitting jobs 329
 - throughput goals 328
 - velocity goals 328

Index

- violation period 339
 - SLA_TIMER parameter in `lsb.params` 352
 - SLOT_POOL parameter, in `lsb.queues` 311
 - SLOT_SHARE parameter in `lsb.queues` 311
 - slots, viewing resource allocation limits (`blimits`) 384
 - soft resource limits
 - data segment size 516
 - description 509
 - example 513
 - file size 517
 - memory usage 517
 - stack segment size 519
 - software licenses
 - counted 261
 - floating
 - dedicated queue for 264
 - description 262
 - host locked 261
 - interactive jobs competing with batch jobs 265
 - managing 265
 - span string 291
 - special characters, defining host names 84
 - specifying, license server TCP port 168
 - spooling. *See* command file spooling, job file spooling
 - SSUSP job state
 - description 111
 - overview 104
 - stack segment size limit 519
 - STACKLIMIT parameter in `lsb.queues` 519
 - standard input and error, splitting for interactive jobs 564
 - standard input and output
 - job arrays 462
 - passing data between `esub` and `eexec` 552
 - standard output and error, redirecting to a file 577
 - Start, service instance property 194
 - started job dependency condition 426
 - State, service instance property 194
 - state, service property 192
 - states, services 191
 - static job priority, absolute job priority scheduling 435
 - static priority fairshare 326
 - static resources
 - See also* individual resource names
 - description 240
 - shared, reserving 253
 - STATUS, `bhosts` 53
 - status
 - closed in `bhosts` 57
 - displaying license server (`lmstat`) 153
 - job arrays 464, 466
 - load index 237
 - viewing, queues 92
 - WAIT for chunk jobs 456
 - stderr and stdout
 - redirecting to a file 577
 - splitting for interactive jobs 564
 - stdin and stdout, passing data between `esub` and `eexec` 552
 - STOP_COND parameter in `lsb.queues` 554
 - stopping, FLEXlm server (`lmdown`) 153
 - string resources 232
 - SUB_TRY_INTERVAL parameter in `lsb.params` 104
 - subfactors, absolute job priority scheduling 434
 - submission executable (`esub`) 544
 - submission options, embedding for interactive jobs 570
 - summary dashboard, overview 186
 - Sun Network Information Service/Yellow Pages. *See* NIS
 - supported file systems 655
 - SUSPEND job control action, default 553
 - suspended jobs
 - resuming 527
 - states 105
 - viewing resource allocation limits (`blimits`) 384
 - suspending conditions
 - configuring 526
 - viewing 526
 - suspending reason, viewing 105, 526
 - suspending thresholds 526
 - `svc.conf` file (name services) 78
 - swap space
 - load index 239
 - suspending conditions 525
 - viewing resource allocation limits (`blimits`) 384
 - SWAPLIMIT parameter in `lsb.queues` 520
 - SWP absolute job priority scheduling factor 434
 - swp load index
 - description 239
 - suspending conditions 525
 - viewing resource allocation limits (`blimits`) 384
 - `syslog.h` file 663
 - system overview 171
- ## T
- task control, with `lstcsh` 712
 - task lists
 - and `lstcsh` 708
 - changing memberships 708
 - tasks
 - file access 577
 - running same on many hosts in sequence 575
 - selecting host to run on 574
 - starting parallel 471
 - TCP service port numbers
 - configuring for NIS or NIS+ databases 76

- registering for LSF 75
- tcsh, version and lscsh 709
- temp space
 - suspending conditions 525
 - viewing resource allocation limits (blimits) 384
- temporary license 136
- TERMINATE job control action 554
- TERMINATE_WHEN parameter in lsb.queues
 - changing default SUSPEND action 557
 - TERMINATE job control action 554
- TerminateProcess() system call (Windows), job control actions 555
- THREADLIMIT parameter in lsb.queues 519
- threads, job limit 519
- thresholds
 - exited job exceptions 88
 - host and queue 31
 - idle job exceptions 100
 - job exit rate for hosts 88, 124
 - job overrun exceptions 100
 - job underrun exceptions 100
 - scheduling and suspending 526
 - tuning for LIM 629
- tilde (~)
 - not operator
 - host partition fairshare 302
 - host-based resources 252
- time expressions
 - creating for automatic configuration 270
 - logical operators 270
- time normalization, CPU factors 521
- time values, specifying 269
- time windows, syntax 269
- time-based configuration
 - commands for checking 273
 - description 271
- time-based resource limits 275
- timing level, commands for daemons 690
- /tmp directory, default LSF_LOGDIR 201, 663
- tmp load index
 - description 239
 - suspending conditions 525
 - viewing resource allocation limits (blimits) 384
- /tmp_mnt directory 672
- troubleshoot
 - hosts 187
 - service error states 195
 - using multiple log files 206
- troubleshooting 166
- Type, service property 193
- type static resource 59, 240
- U**
- UDP service port numbers, registering for LSF 75
- UJOB_LIMIT parameter in lsb.queues 377
- unavail host status
 - bhosts command 53
 - lslod command 55
 - status load index, status load index 237
- uncondensed host groups, viewing 57
- underrun job exceptions
 - configuring 100
 - description 100, 123
 - viewing with bjobs 108
 - viewing with bqueues 94
- UNICOS accounting 662
- UNIX directory structure, example 43
- UNIX/Windows user account mapping, scope 212
- unlicensed cluster 141
- unlicensed host status
 - bhosts command 54
 - lslod command 55
 - status load index 237
- unreach host status, bhosts command 54
- update interval 665
 - duplicate event logging 665
- updating a license 149
- usage limits. *See* resource usage limits
- usage string 287
- user account, modify 185
- user accounts
 - about 172, 198
 - adding a consumer administrator 199
- user authentication, security 642
- user groups
 - defining external user groups 133
 - overview 131
 - specifying 323
 - viewing information about 129
- user home directories, shared 32
- user priority
 - description 298
 - formula 299
- user roles, description 172, 199
- user share assignments 297
- USER_ADVANCE_RESERVATION parameter in lsb.params, obsolete parameter 406
- USER_NAME parameter in lsb.users 132
- USER_NAME parameter in lsb.users file 132
- USER_SHARES parameter in lsb.hosts 132
- USER_SHARES parameter in lsb.hosts file 132
- user-based host partition fairshare, resource usage measurement 299
- user-based queue-level fairshare, resource usage measurement 299
- users
 - viewing information about 129
 - viewing jobs submitted by 106

Index

- viewing resource allocation limits (blimits) 384
- viewing shares 130
- USERS parameter in lsb.queues 132
- USERS parameter in lsb.queues file 132
- user-specified job requeue 443
- /usr/include/sys/syslog.h file 663
- %USRCMD string in job starters 540
- USUSP job state
 - description 111
 - overview 104
 - suspending and resuming jobs 111
- ut load index, built-in resource 238
- utmp file registration on IRIX, enabling 571
- V**
- variables. *See* individual environment variable names
- vendor daemon (lsf_ld) 140
- verifying LSF floating client 164
- version information, displaying in FLEXlm (lmver) 153
- viewing, configuration errors 52
- viewing condensed and uncondensed 57
- violation period, SLA scheduling 339
- virtual memory
 - load index 239
 - suspending conditions 525
- virtual memory limit 520
- vmstat 239
- W**
- WAIT status of chunk jobs
 - description 456
 - viewing 107
- wall-clock run time limit 519
- web service gateway, about 191
- WEBGUI service, about 191
- weekly planner for advance reservation (brsvs-p) 411
- wildcards, defining host names 84
- Windows
 - default directory structure 44
 - job control actions 555
 - TerminateProcess() system call, job control actions 555
- windows
 - dispatch 420
 - run 419
 - time 269
- Windows Event Viewer 623
- workarounds to lsrcp limitations 658
- X**
- X applications, running with bsub 568
- xterm, starting in LSF Base 578
- Y**
- yplib daemon 79
- ypcat hosts.byname 79
- ypmake command 77