Running Jobs with Platform LSF™

Version 7 Update 2
Release date: November 2007
Last modified: December 3, 2007
Support: support@platform.com
Comments to: doc@platform.com

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.

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.

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.

PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOBSCHEDULER, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, PLATFORM VM ORCHESTRATOR, PLATFORM VMO, ACCELERATING INTELLIGENCE, 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.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

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.

Macrovision, Globetrotter and FLEX*lm* are registered trademarks or trademarks of Macrovision Corporation in the United States of America and/or other countries.

Topspin is a registered trademark of Topspin Communications, Inc.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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

www.platform.com/Company/third.part.license.htm

Third Party Copyright Notices

www.platform.com/Company/Third.Party.Copyright.htm

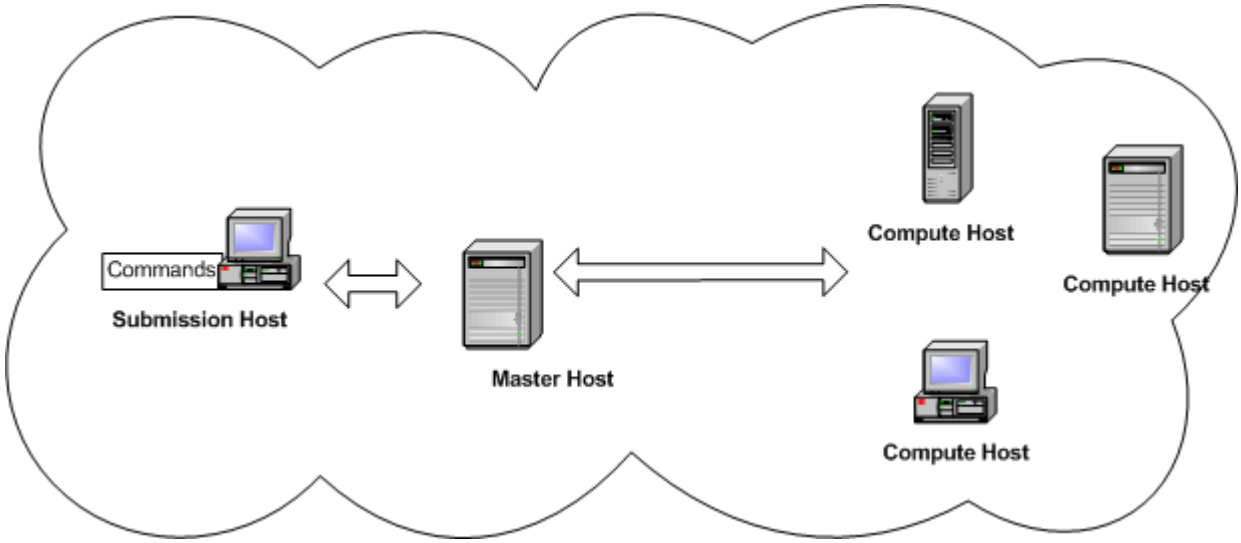
Contents

CHAPTER 1

About Platform LSF

Contents

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`*

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
- `bparams`—View default queues

Configuration

- Define queues in `lsb.queues`

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

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 the first host listed in the `lsf.cluster.`*`cluster_name`* file or is defined along with other candidate master hosts by LSF_MASTER_LIST in `lsf.conf`.

LSF daemons

LSF daemon	Role
`mbatchd`	Job requests and dispatch
`mbschd`	Job scheduling
`sbatchd` `res`	Job execution

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)

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

- `lsrun`—Submit an interactive task
- `lsgrun`—Submit an interactive task to a group of hosts
- See also LSF utilities such as `ch`, `lsacct`, `lsacctmrg`, `lslogin`, `lsplace`, `lsload`, `lsloadadj`, `lseligible`, `lsmon`, `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
- `.lsftasks`—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

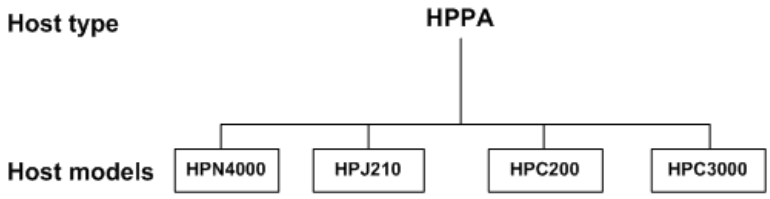
Configuration

- `lsf.task`—Configure systemwide resource requirements for tasks
- `lsf.task.cluster`—Configure clusterwide resource requirements for tasks
- `.lsftasks`—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 `lsf.cluster.cluster_name`. 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

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.

For example, a cluster administrator can create an LSF host group, submit a job to any queue, or terminate another user's job.

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

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.

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.

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 PIM.

Commands

- `lsinfo`—View the resources available in your cluster
- `bjobs -l`—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

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.

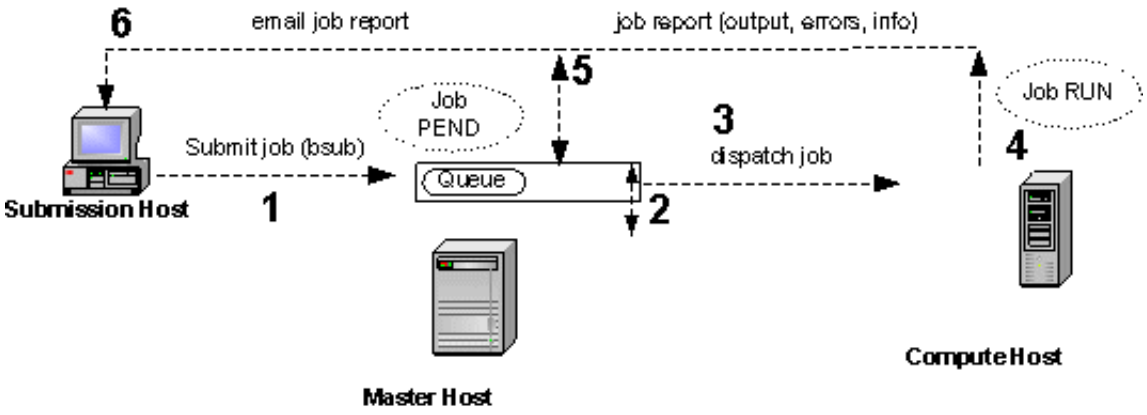
Commands

- `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 PEND state. The job is held in a job file in the `LSF_SHAREDIR/`*`cluster_name`*`/logdir/info/` directory.

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

CHAPTER

2

Working with Jobs

Contents

Submitting Jobs (bsub)

In this section

bsub command

You submit a job with the `bsub` command. If you do not specify any options, the job is submitted to the default queue configured by the LSF administrator (usually queue `normal`).

For example, if you submit the job `my_job` without specifying a queue, the job goes to the default queue.

```
bsub my_job
Job <1234> is submitted to default queue <normal>
```

In the above example, 1234 is the job ID assigned to this job, and `normal` is the name of the default job queue.

Your job remains pending until all conditions for its execution are met. Each queue has execution conditions that apply to all jobs in the queue, and you can specify additional conditions when you submit the job.

You can also specify an execution host or a range of hosts, a queue, and start and termination times, as well as a wide range of other job options. See the `bsub` command in the *Platform LSF Command Reference* for more details on `bsub` options.

Submitting a job to a specific queue (bsub -q)

Job queues represent different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Each job queue can use a configured subset of server hosts in the cluster; the default is to use all server hosts.

System administrators can configure job queues to control resource access by different users and types of application. Users select the job queue that best fits each job.

The default queue is normally suitable to run most jobs, but the default queue may assign your jobs a very low priority, or restrict execution conditions to minimize interference with other jobs. If automatic queue selection is not satisfactory, choose the most suitable queue for each job.

The factors affecting which queue to choose are user access restrictions, size of job, resource limits of the queue, scheduling priority of the queue, active time windows of the queue, hosts used by the queue, scheduling load conditions, and the queue description displayed by the `bqueues -l` command.

Viewing available queues

To see available queues, use the `bqueues` command.

Use `bqueues -u` *`user_name`* to specify a user or user group so that `bqueues` displays only the queues that accept jobs from these users.

The `bqueues -m` *`host_name`* option allows users to specify a host name or host group name so that `bqueues` displays only the queues that use these hosts to run jobs.

You can submit jobs to a queue as long as its `STATUS` is `Open`. However, jobs are not dispatched unless the queue is `Active`.

Submitting a job

The following examples are based on the queues defined in the default configuration. Your LSF administrator may have configured different queues.

To run a job during off hours because the job generates very high load to both the file server and the network, you can submit it to the night queue:

```
bsub -q night
```

If you have an urgent job to run, you may want to submit it to the priority queue:

```
bsub -q priority
```

If you want to use hosts owned by others and you do not want to bother the owners, you may want to run your low priority jobs on the idle queue so that as soon as the owner comes back, your jobs get suspended:

```
bsub -q idle
```

If you are running small jobs and do not want to wait too long to get the results, you can submit jobs to the short queue to be dispatched with higher priority:

```
bsub -q short
```

Make sure your jobs are short enough that they are not killed for exceeding the CPU time limit of the queue (check the resource limits of the queue, if any).

If your job requires a specific execution environment, you may need to submit it to a queue that has a particular job starter defined. LSF administrators are able to specify a queue-level job starter as part of the queue definition; ask them for the name of the queue and configuration details.

See *Administering Platform LSF* for information on queue-level job starters.

Submitting a job associated to a project (bsub -P)

Use the `bsub -P` *`project_name`* option to associate a project name with a job.

Project names are logged in `lsb.acct`. You can use the `bacct` command to gather accounting information on a per-project basis.

On systems running IRIX 6, before the submitted job begins execution, a new array session is created and the project ID corresponding to the project name is assigned to the session.

Submitting a job associated to a user group (bsub -G)

You can use the `bsub -G user_group` option to submit a job and associate it with a specified user group. This option is only useful with fairshare scheduling.

For more details on fairshare scheduling, see *Administering Platform LSF*.

You can specify any user group to which you belong as long as it does not contain any subgroups. You must be a direct member of the specified user group.

User groups in non-leaf nodes cannot be specified because it will cause ambiguity in determining the correct shares given to a user.

For example, to submit the job `myjob` associated to user group `special`:

```
bsub -G special myjob
```

Submitting a job with a job name (bsub -J)

Use `bsub -J job_name` to submit a job and assign a job name to it.

Job names can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows.

You can later use the job name to identify the job. The job name need not be unique.

For example, to submit a job and assign the name `my_job`:

```
bsub -J my_job
```

You can also assign a job name to a job array. See *Administering Platform LSF* for more information about job arrays.

Submitting a job to a service class (bsub -sla)

Use the `bsub -sla service_class_name` to submit a job to a service class for SLA-driven scheduling.

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.

For example:

```
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`.

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`.

You should submit your jobs with a run time limit (`-W` option) or the queue should specify a run time limit (RUNLIMIT in the queue definition in `lsb.queues`). If you do not specify a run time limit, LSF automatically adjusts the optimum number of running jobs according to the observed run time of finished jobs.

See *Administering Platform LSF* for more information about service classes and goal-oriented SLA driven scheduling.

Submitting a job under a job group (bsub -g)

Use `bsub -g` to submit a job into a job group. The job group does not have to exist before submitting the job. For example:

```
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.

See *Administering Platform LSF* for more information about job groups.

Modifying a Submitted Job (bmod)

In this section

Modifying Pending Jobs (bmod)

If your submitted jobs are pending (`bjobs` shows the job in PEND state), use the `bmod` command to modify job submission parameters. You can also modify entire job arrays or individual elements of a job array.

See the `bmod` command in the *Platform LSF Command Reference* for more details.

Replacing the job command–line

To replace the job command line, use the `bmod -Z "new_command"` option. The following example replaces the command line option for job 101 with `"myjob file"`:

```
bmod -Z "myjob file" 101
```

Changing a job parameter

To change a specific job parameter, use `bmod` with the `bsub` option used to specify the parameter. The specified options replace the submitted options. The following example changes the start time of job 101 to 2:00 a.m.:

```
bmod -b 2:00 101
```

Resetting to default submitted value

To reset an option to its default submitted value (undo a `bmod`), append the `n` character to the option name, and do not include an option value. The following example resets the start time for job 101 back to its default value:

```
bmod -bn 101
```

Resource reservation can be modified after a job has been started to ensure proper reservation and optimal resource utilization.

Modifying a job submitted to a service class

Use the `-sla` option of `bmod` to modify the service class a job is attached to, or to attach a submitted job to a service class. Use `bmod -slan` to detach a job from a service class. For example:

```
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

See *Administering Platform LSF* for more information about submitting jobs to service classes for SLA-driven scheduling.

Modifying a job submitted to a job group

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. For example:

`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. 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;
Wed May 14 15:24:17: Dispatched to <hostA>;
Wed May 14 15:24:17: Starting (Pid 8602);
...
```

See *Administering Platform LSF* for more information about job groups.

Modifying Running Jobs

Modifying resource reservation

A job is usually submitted with a resource reservation for the maximum amount required. Use `bmod -R` to modify the resource reservation for a running job. This command is usually used to decrease the reservation, allowing other jobs access to the resource.

The following example sets the resource reservation for job 101 to 25MB of memory and 50 MB of swap space:

```
bmod -R "rusage[mem=25:swp=50]" 101
```

By default, you can modify resource reservation for running jobs. Set LSB_MOD_ALL_JOBS in `lsf.conf` to modify additional job options.

See "Reserving Resources for Jobs" on page 40 for more details.

Modifying other job options

If LSB_MOD_ALL_JOBS is specified in `lsf.conf`, the job owner or the LSF administrator can use the `bmod` command to modify the following job options for running jobs:

- CPU limit (`-c` [*hour*`:`]*minute*[`/`*host_name* | `/`*host_model*] | `-cn`)
- Memory limit (`-M` *mem_limit* | `-Mn`)
- Run limit (`-W` *run_limit*[`/`*host_name* | `/`*host_model*] | `-Wn`)
- Standard output file name (`-o` *output_file* | `-on`)
- Standard error file name (`-e` *error_file* | `-en`)
- Rerunnable jobs (`-r` | `-rn`)

In addition to resource reservation, these are the only `bmod` options that are valid for running jobs. You cannot make any other modifications after a job has been dispatched.

An error message is issued and the modification fails if these options are used on running jobs in combination with other `bmod` options.

Modifying resource limits for running jobs

The new resource limits cannot exceed the resource limits defined in the queue.

To modify the CPU limit of running jobs, LSB_JOB_CPULIMIT=Y must be defined in `lsf.conf`.

To modify the memory limit of running jobs, LSB_JOB_MEMLIMIT=Y must be defined in `lsf.conf`.

Limitations

Modifying remote running jobs in a MultiCluster environment is not supported.

To modify the name of job error file for a running job, you must use `bsub -e` or `bmod -e` to specify an error file before the job starts running.

For more information

See *Administering Platform LSF* for more information about job output files, using job-level resource limits, and submitting rerunnable jobs.

Controlling Jobs

LSF controls jobs dispatched to a host to enforce scheduling policies, or in response to user requests. The LSF system performs the following actions on a job:

- Suspend by sending a `SIGSTOP` signal
- Resume by sending a `SIGCONT` signal
- Terminate by sending a `SIGKILL` signal

On Windows, equivalent functions have been implemented to perform the same tasks.

In this section

Killing Jobs (bkill)

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`, which waits for the job to exit before reporting the status. Because of these delays, for a short period of time after entering the `bkill` command, `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.

Example

To kill job 3421:

```
bkill 3421
Job <3421> is being terminated
```

Forcing removal of a job from LSF

If a job cannot be killed in the operating system, use `bkill -r` to force the removal of the job from LSF.

The `bkill -r` command removes a job from the 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.

Suspending and Resuming Jobs (bstop and bresume)

The `bstop` and `bresume` commands allow you to suspend or resume a job.

A job can also be suspended by its owner or the LSF administrator with the `bstop` command. These jobs are considered user-suspended and are displayed by `bjobs` as `USUSP`.

When the user restarts the job with the `bresume` command, the job is not started immediately to prevent overloading. Instead, the job is changed from `USUSP` to `SSUSP` (suspended by the system). The `SSUSP` job is resumed when host load levels are within the scheduling thresholds for that job, similarly to jobs suspended due to high load.

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. See *Administering Platform LSF* for information about configuring queues.

Suspending a job

bstop command

To suspend a job, use the `bstop` command. Suspending a job causes your job to go into `USUSP` state if the job is already started, or to go into `PSUSP` state if your job is pending.

By default, jobs that are suspended by the administrator can only be resumed by the administrator or `root`; users do not have permission to resume a job suspended by another user or the administrator. Administrators can resume jobs suspended by users or administrators. Administrators can also enable users to resume their own jobs that have been stopped by an administrator.

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`.

Example

To suspend job 3421, enter:

```
bstop 3421
Job <3421> is being stopped
```

Resuming a job

bresume command

To resume a job, use the `bresume` command.

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.

For example, to resume job 3421, enter:

```
bresume 3421
Job <3421> is being resumed
```

You cannot resume jobs suspended by another user; you can only resume your own jobs. If your job was suspended by the administrator, you cannot resume it; the administrator or `root` must resume the job for you.

ENABLE_USER_RESUME parameter (lsb.params)

If ENABLE_USER_RESUME=Y in `lsb.params`, you can resume your own jobs that have been suspended by the administrator.

Changing Job Order Within Queues (bbot and btop)

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.

Moving a job to the bottom of a queue

Use `bbot` to move 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.

Moving a job to the top of a queue

Use `btop` to move 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.

Example

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
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
```

Controlling Jobs in Job Groups

Stopping (bstop)

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
```

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
```

Resuming (bresume)

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
```

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
```

Terminating (bkill)

Use the -g option of bkill and specify a job group path to terminate jobs in a job group. For example,

```
bkill -g /risk_group 106
Job <106> is being terminated
```

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
```

Deleting (bgdel)

Use `bgdel` command to remove a job group. The job group cannot contain any jobs. For example:

```
bgdel /risk_group
Job group /risk_group is deleted.
```

deletes the job group `/risk_group` and all its subgroups.

For more information

See *Administering Platform LSF* for more information about using job groups.

Submitting a Job to Specific Hosts

To indicate that a job must run on one of the specified hosts, use the `bsub -m "hostA hostB ..."` option.

By specifying a single host, you can force your job to wait until that host is available and then run on that host.

For example:

```
bsub -q idle -m "hostA hostD hostB" myjob
```

This command submits `myjob` to the idle queue and tells LSF to choose one host from `hostA`, `hostD` and `hostB` to run the job. All other batch scheduling conditions still apply, so the selected host must be eligible to run the job.

Resources and bsub -m

If you have applications that need specific resources, it is more flexible to create a new Boolean resource and configure that resource for the appropriate hosts in the cluster.

This must be done by the LSF administrator. If you specify a host list using the `-m` option of `bsub`, you must change the host list every time you add a new host that supports the desired resources. By using a Boolean resource, the LSF administrator can add, move or remove resources without forcing users to learn about changes to resource configuration.

Submitting a Job and Indicating Host Preference

When several hosts can satisfy the resource requirements of a job, the hosts are ordered by load. However, in certain situations it may be desirable to override this behavior to give preference to specific hosts, even if they are more heavily loaded.

For example, you may have licensed software which runs on different groups of hosts, but you prefer it to run on a particular host group because the jobs will finish faster, thereby freeing the software license to be used by other jobs.

Another situation arises in clusters consisting of dedicated batch servers and desktop machines which can also run jobs when no user is logged in. You may prefer to run on the batch servers and only use the desktop machines if no server is available.

To see a list of available hosts, use the `bhosts` command.

In this section

Submitting a job with host preference

bsub -m

The `bsub -m` option allows you to indicate preference by using + with an optional preference level after the host name. The keyword `others` can be used to refer to all the hosts that are not explicitly listed. You must specify `others` with at least one host name or host group name.

For example:

```
bsub -m "hostD+ others" -R "solaris && mem> 10" myjob
```

In this example, LSF selects all `solaris` hosts that have more than 10 MB of memory available. If `hostD` meets this criteria, it will be picked over any other host which otherwise meets the same criteria. If `hostD` does not meet the criteria, the least loaded host among the others will be selected. All the other hosts are considered as a group and are ordered by load.

Queues and host preference

A queue can also define host preferences for jobs. Host preferences specified by `bsub -m` override the queue specification.

In the queue definition in `lsb.queues`, use the HOSTS parameter to list the hosts or host groups to which the queue can dispatch jobs.

Use the not operator (~) to exclude hosts or host groups from the list of hosts to which the queue can dispatch jobs. This is useful if you have a large cluster, but only want to exclude a few hosts from the queue definition.

See the *Platform LSF Reference* for information about the `lsb.queues` file.

Submitting a job with different levels of host preference

You can indicate different levels of preference by specifying a number after the plus sign (+). The larger the number, the higher the preference for that host or host group. You can also specify the + with the keyword `others`.

For example:

```
bsub -m "groupA+2 groupB+1 groupC" myjob
```

In this example, LSF gives first preference to hosts in `groupA`, second preference to hosts in `groupB` and last preference to those in `groupC`. Ordering within a group is still determined by load.

You can use the `bmgroup` command to display configured host groups.

Submitting a job with resource requirements

To submit a job which will run on Solaris 7 or Solaris 8:

```
bsub -R "sol7 || sol8" myjob
```

When you submit a job, you can also exclude a host by specifying a resource requirement using hname resource:

```
bsub -R "hname!=hostb && type==sgi6" myjob
```

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 the 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 *Administering Platform LSF* for more information about file transfer in LSF.

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.

Reserving Resources for Jobs

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.

You can reserve resources 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.

Viewing host-level resource information

Use `bhosts -l` to view the amount of resources reserved on each host. Use `bhosts -s` to view information about shared resources.

Viewing queue-level resource information

To see the resource usage configured at the queue level, use `bqueues -l`.

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 the rusage string

To specify resource reservation at the job level, use `bsub -R` and include the resource usage section in the resource requirement (`rusage`) string.

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.

Submitting a Job with Start or Termination Times

By default, LSF dispatches jobs as soon as possible, and then allows them to finish, although resource limits might terminate the job before it finishes.

You can specify a time of day at which to start or terminate a job.

Submitting a job with a start time

If you do not want to start your job immediately when you submit it, use `bsub -b` to specify a start time. LSF will not dispatch the job before this time. For example:

```
bsub -b 5:00 myjob
```

This example submits a job that remains pending until after the local time on the master host reaches 5 a.m.

Submitting a job with a termination time

Use `bsub -t` to submit a job and specify a time after which the job should be terminated. For example:

```
bsub -b 11:12:5:40 -t 11:12:20:30 myjob
```

The job called `myjob` is submitted to the default queue and will start after November 12 at 05:40 a.m. If the job is still running on November 12 at 8:30 p.m., it will be killed.

CHAPTER

3

Viewing Information About Jobs

Use the `bjobs` and `bhist` commands to view information about jobs:

- `bjobs` reports the status of jobs and the various options allow you to display specific information.
- `bhist` reports the history of one or more jobs in the system.

You can also find jobs on specific queues or hosts, find jobs submitted by specific projects, and check the status of specific jobs using their job IDs or names.

Contents

Viewing Job Information (bjobs)

The `bjobs` command has options to display the status of jobs in the LSF system. For more details on these or other `bjobs` options, see the `bjobs` command in the *Platform LSF Command Reference*.

Unfinished current jobs

The `bjobs` command reports the status of LSF jobs.

When no options are specified, `bjobs` displays information about jobs in the PEND, RUN, USUSP, PSUSP, and SSUSP states for the current user.

For example:

bjobs

```
JOBID USER   STAT   QUEUE     FROM_HOST EXEC_HOST JOB_NAME    SUBMIT_TIME
3926  user1  RUN    priority  hostf     hostc     verilog     Oct 22 13:51
605   user1  SSUSP  idle      hostq     hostc     Test4       Oct 17 18:07
1480  user1  PEND   priority  hostd               generator   Oct 19 18:13
7678  user1  PEND   priority  hostd               verilog     Oct 28 13:08
7679  user1  PEND   priority  hosta               coreHunter  Oct 28 13:12
7680  user1  PEND   priority  hostb               myjob       Oct 28 13:17
```

All jobs

`bjobs -a` displays the same information as `bjobs` and in addition displays information about recently finished jobs (PEND, RUN, USUSP, PSUSP, SSUSP, DONE and EXIT statuses).

All your jobs that are still in the system and jobs that have recently finished are displayed.

Running jobs

`bjobs -r` displays information only for running jobs (RUN state).

Viewing Job Pend and Suspend Reasons (bjobs -p)

When you submit a job, it may be held in the queue before it starts running and it may be suspended while running. You can find out why jobs are pending or in suspension with the `bjobs -p` option.

You can combine `bjob` options to tailor the output. For more details on these or other `bjobs` options, see the `bjobs` command in the *Platform LSF Command Reference*.

In this section

Pending jobs and reasons

`bjobs -p` displays information for pending jobs (PEND state) and their reasons. There can be more than one reason why the job is pending.

For example:

```
bjobs -p
JOBID USER   STAT   QUEUE    FROM_HOST    JOB_NAME    SUBMIT_TIME
7678  user1  PEND   priority  hostD       verilog      Oct 28 13:08
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;
```

The pending reasons also mention the number of hosts for each condition.

You can view reasons why a job is pending or in suspension for all users by combining the `-p` and `-u all` options.

Viewing pending and suspend reasons with host names

To get specific host names along with pending reasons, use the `-p` and `-l` options with the `bjobs` command.

For example:

```
bjobs -lp
Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>
, Command <verilog>
Mon Oct 28 13:08:11: Submitted from host <hostD>,CWD <$HOME>, Requested
Resources <type==any && swp>35>;

PENDING REASONS:
Queue's resource requirements not satisfied: hostb, hostk, hostv;
Unable to reach slave lsbatch server: hostH;
Not enough job slots: hostF;

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

Viewing suspend reasons only

The -s option of bjobs displays reasons for suspended jobs only. For example:

```
bjobs -s
JOBID USER  STAT  QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
605   user1 SSUSP idle  hosta     hostc     Test4    Oct 17 18:07
The host load exceeded the following threshold(s):
Paging rate: pg;
Idle time: it;
```

Viewing Detailed Job Information (bjobs -l)

The `-l` option of `bjobs` displays detailed information about job status and parameters, such as the job's current working directory, parameters specified when the job was submitted, and the time when the job started running. For more details on `bjobs` options, see the `bjobs` command in the *Platform LSF Command Reference*.

`bjobs -l` with a job ID displays all the information about a job, including:

- Submission parameters
- Execution environment
- Resource usage

For example:

```
bjobs -l 7678
Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>
, Command <verilog>
Mon Oct 28 13:08:11: Submitted from host <hostD>,CWD <$HOME>,
Requested Resources <type==any && swp>35>;
PENDING REASONS:
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;

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

Viewing Job Resource Usage (bjobs -l)

LSF monitors the resources jobs consume while they are running. The `-l` option of the `bjobs` command displays the current resource usage of the job.

For more details on `bjobs` options, see the `bjobs` command in the *Platform LSF Command Reference*.

Job-level information

Job-level information includes:

- Total CPU time consumed by all processes of a job
- Total resident memory usage in KB of all currently running processes of a job
- Total virtual memory usage in KB of all currently running processes of a job
- Currently active process group ID of a job
- Currently active processes of a job

Update interval

The job-level resource usage information is updated at a maximum frequency of every SBD_SLEEP_TIME seconds. See the *Platform LSF Command Reference* for the value of SBD_SLEEP_TIME.

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.

Viewing job resource usage

To view resource usage for a specific job, specify `bjobs -l` with the job ID:

```
bjobs -l 1531
Job Id <1531>, User <user1>, Project <default>, Status <RUN>, Queue
<priority> Command <example 200>
Fri Dec 27 13:04:14   Submitted from host <hostA>, CWD <$HOME>,
SpecifiedHosts <hostD>;
Fri Dec 27 13:04:19: Started on <hostD>, Execution Home </home/user1>, Executio
n CWD </home/user1>;
Fri Dec 27 13:05:00: Resource usage collected.
The CPU time used is 2 seconds.
MEM: 147 Kbytes; SWAP: 201 Kbytes PGID: 8920;  PIDs: 8920 8921 8922

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

Viewing Job History (bhist)

Sometimes you want to know what has happened to your job since it was submitted. The `bhist` command displays a summary of the pending, suspended and running time of jobs for the user who invoked the command. Use `bhist -u all` to display a summary for all users in the cluster.

For more details on `bhist` options, see the `bhist` command in the *Platform LSF Command Reference*.

In this section

Viewing detailed job history

The `-l` option of `bhist` displays the time information and a complete history of scheduling events for each job.

```
bhist -l 1531
JobId <1531>, User <user1>, Project <default>, Command< example200>
Fri Dec 27 13:04:14: Submitted from host <hostA> to Queue <priority>,
CWD <$HOME>, Specified Hosts <hostD>;
Fri Dec 27 13:04:19: Dispatched to <hostD>;
Fri Dec 27 13:04:19: Starting (Pid 8920);
Fri Dec 27 13:04:20: Running with execution home </home/user1>, Execution CWD
</home/user1>, Execution Pid <8920>;
Fri Dec 27 13:05:49: Suspended by the user or administrator;
Fri Dec 27 13:05:56: Suspended: Waiting for re-scheduling after being resumed
by user;
Fri Dec 27 13:05:57: Running;
Fri Dec 27 13:07:52: Done successfully. The CPU time used is 28.3 seconds.

Summary of time in seconds spent in various states by Sat Dec 27 13:07:52 1997
PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
5     0      205  7      1      0      218
```

Viewing history of jobs not listed in active event log

LSF periodically backs up and prunes the job history log. By default, `bhist` only displays job history from the current event log file. You can use `bhist -n num_logfiles` to display the history for jobs that completed some time ago and are no longer listed in the active event log.

bhist -n *num_logfiles*

The `-n num_logfiles` option tells the `bhist` command to search through the specified number of log files instead of only searching the current log file.

Log files are searched in reverse time order. For example, the command `bhist -n 3` searches the current event log file and then the two most recent backup files.

Examples

`bhist -n 1`	searches the current event log file `lsb.events`
`bhist -n 2`	searches `lsb.events` and `lsb.events.1`
`bhist -n 3`	searches `lsb.events`, `lsb.events.1`, `lsb.events.2`
`bhist -n 0`	searches all event log files in LSB_SHAREDIR

Viewing chronological history of jobs

By default, the `bhist` command displays information from the job event history file, `lsb.events`, on a per job basis.

bhist -t

The `-t` option of `bhist` can be used to display the events chronologically instead of grouping all events for each job.

bhist -T

The `-T` option allows you to select only those events within a given time range.

For example, the following displays all events which occurred between 14:00 and 14:30 on a given day:

```
bhist -t -T 14:00,14:30
Wed Oct 22 14:01:25: Job <1574> done successfully;
Wed Oct 22 14:03:09: Job <1575> submitted from host to Queue , CWD , User ,
Project , Command , Requested Resources ;
Wed Oct 22 14:03:18: Job <1575> dispatched to ;
Wed Oct 22 14:03:18: Job <1575> starting (Pid 210);
Wed Oct 22 14:03:18: Job <1575> running with execution home , Execution CWD ,
Execution Pid <210>;
Wed Oct 22 14:05:06: Job <1577> submitted from host  to Queue, CWD , User ,
Project , Command , Requested Resources ;
Wed Oct 22 14:05:11: Job <1577> dispatched to ;
Wed Oct 22 14:05:11: Job <1577> starting (Pid 429);
Wed Oct 22 14:05:12: Job <1577> running with execution home, Execution CWD ,
Execution Pid <429>;
Wed Oct 22 14:08:26: Job <1578> submitted from host to Queue, CWD , User ,
Project , Command;
Wed Oct 22 14:10:55: Job <1577> done successfully;
Wed Oct 22 14:16:55: Job <1578> exited;
Wed Oct 22 14:17:04: Job <1575> done successfully;
```

Viewing Job Output (bpeek)

The output from a job is normally not available until the job is finished. However, LSF provides the `bpeek` command for you to look at the output the job has produced so far.

By default, `bpeek` shows the output from the most recently submitted job. You can also select the job by queue or execution host, or specify the job ID or job name on the command line.

For more details on `bpeek` options, see the `bpeek` command in the *Platform LSF Reference*.

Viewing output of a running job

Only the job owner can use `bpeek` to see job output. The `bpeek` command will not work on a job running under a different user account.

To save time, you can use this command to check if your job is behaving as you expected and kill the job if it is running away or producing unusable results.

For example:

```
bpeek 1234
<< output from stdout >>
Starting phase 1
Phase 1 done
Calculating new parameters
...
```

Viewing Information about SLAs and Service Classes

Monitoring the progress of an SLA (bsla)

Use `bsla` to display the properties of service classes configured in `lsb.serviceclasses` and dynamic state information for each service class.

Examples

- One velocity goal of service class `Tofino` is active and on time. The other configured velocity goal is inactive.

```
bsla Tofino
SERVICE CLASS NAME: Tofino
 -- day and night velocity
PRIORITY: 20

GOAL: VELOCITY
ACTIVE WINDOW: (9:00-17:00)
STATUS: Active:On time
VELOCITY:  10
CURRENT VELOCITY:  10

GOAL:  VELOCITY
ACTIVE WINDOW: (17:30-8:30)
STATUS:  Inactive
VELOCITY:  30
CURRENT VELOCITY:  0

   NJOBS    PEND     RUN      SSUSP    USUSP    FINISH
     360     300      10          2        0        48
```

- The deadline goal of service class `Uclulet` is not being met, and `bsla` displays status `Active:Delayed`:

```
bsla Uclulet
SERVICE CLASS NAME:  Uclulet
 -- working hours
PRIORITY: 20

GOAL:  DEADLINE
ACTIVE WINDOW: (8:30-16:00)
DEADLINE:  (Tue Jun 24 16:00)
ESTIMATED FINISH TIME:  (Wed Jun 25 14:30)
OPTIMUM NUMBER OF RUNNING JOBS:  2
STATUS:  Active:Delayed

   NJOBS    PEND     RUN      SSUSP    USUSP    FINISH
      13       0       0          0        0        13
```

- 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
ACTIVE WINDOW: (9:00-17:30)
STATUS:  Active:On time
VELOCITY:  8
CURRENT VELOCITY:  0

GOAL:  DEADLINE
ACTIVE WINDOW: (17:30-9:00)
STATUS:  Inactive

   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
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBs/CLEAN_PERIOD
THROUGHPUT:  6
OPTIMUM NUMBER OF RUNNING JOBS:  5

   NJOBS    PEND     RUN      SSUSP    USUSP    FINISH
    110      95       5          0        0        10
```

Tracking historical behavior of an SLA (bacct)

Use `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
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBs/CLEAN_PERIOD
THROUGHPUT:  6
```

```
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
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  4.00 JOBs/CLEAN_PERIOD
THROUGHPUT:  3
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.

For more information

See *Administering Platform LSF* for more information about service classes and goal-oriented SLA driven scheduling.

Viewing Jobs in Job Groups

Viewing job group information (bjgroup)

Use the `bjgroup` command to see information about jobs in job groups.

```
bjgroup
GROUP_NAME      NJOBS    PEND     RUN    SSUSP  USUSP  FINISH     SLA  JLIMIT  OWNER
/fund1_grp          5       4       0        1      0       0  Inuvik     1/-   user1
/fund2_grp         11       2       5        0      0       4      ()     5/-   user1
/bond_grp           2       2       0        0      0       0      ()     0/-   user2
/risk_grp           2       1       1        0      0       0      ()     1/-   user3
/admi_grp           4       4       0        0      0       0      ()     0/-   user3
```

Viewing jobs by job group (bjobs)

Use the `-g` option of `bjobs` 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                   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>;
...
```

For more information

See *Administering Platform LSF* for more information about using job groups.

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)

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 maximum memory (`maxmem`) of 249 MB, and MEM is limited to 10% of available memory. If 10 MB out of 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 builtin resources (slots, mem, tmp, and swp load indices) 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
```

`blimits` displays the following:

`blimits`

```
INTERNAL RESOURCE LIMITS:

NAME    USERS QUEUES    HOSTS          PROJECTS    SLOTS     MEM       TMP       SWP
limit1  user1    q2  hostA@cluster1         -        -    10/25         -    10/258
limit1  user1    q3  hostA@cluster1         -        -        -    30/2953         -
limit1  user1    q4  hostC                  -        -        -     40/590         -

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.
- 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.

Use `lsload` and `lshosts` to confirm the percentage display:

`lsload`

```
HOST_NAME   status  r15s   r1m  r15m   ut    pg  ls    it   tmp   swp   mem
hostA           ok   0.0   0.0   0.0   0%   2.8   2     0 7046M  515M  174M
hostC           ok   0.0   0.0   0.2   0%   2.1   2     1  745M  133M  170M
hostB           ok   0.0   0.0   0.0   0%   4.5   2   116 6332M  469M  129M
```

`lshosts`

```
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
hostB       LINUX86   PC1133  23.1     1   249M   509M    Yes ()
hostA       LINUX86   PC1133  23.1     1   248M   517M    Yes ()
hostC       LINUX86   PC1133  23.1     1   500M   133M    Yes ()
```

Index