Administering Platform **LSF®**

Version 6.0
January 2004
Comments to: doc@platform.com

Platform™

Last update: January 8 2004

Latest version: www.platform.com/services/support/docs_home.asp

Contents

Welcome

Contents

About This Guide

Purpose of this guide

This guide describes how to manage and configure Platform **LSF**® software ("LSF"). In it, you will find information to do the following:

- Configure and maintain your cluster
- Configure and manage queues, hosts, and users
- Run jobs and control job execution
- Understand and work with resources
- Understand and configure scheduling policies
- Manage job scheduling and dispatch

Who should use this guide

This guide is intended for Platform LSF cluster administrators who need to implement business policies in LSF. Users who want more in-depth understanding of advanced details of LSF operation should also read this guide. Users who simply want to run and monitor their jobs should read *Running Jobs with Platform LSF*.

What you should already know

This guide assumes:

- You have knowledge of system administration tasks such as creating user accounts, sharing and mounting Network File System (NFS) partitions, and backing up the system
- You are familiar with basic LSF concepts and basic LSF operations

Typographical conventions

Typeface	Meaning	Example
`Courier`	The names of on-screen computer output, commands, files, and directories	The `lsid` command
`Bold Courier`	What you type, exactly as shown	Type **`cd /bin`**
Italics	◆ Book titles, new words or terms, or words to be emphasized ◆ Command-line place holders—replace with a real name or value	The queue specified by *queue_name*
Bold Sans Serif	◆ Names of GUI elements that you manipulate	Click **OK**

Command notation

Notation	Meaning	Example
Quotes " or '	Must be entered exactly as shown	"*job_ID*[*index_list*]"
Commas ,	Must be entered exactly as shown	-C *time0*,*time1*
Ellipsis …	The argument before the ellipsis can be repeated. Do not enter the ellipsis.	*job_ID …*
lower case italics	The argument must be replaced with a real value you provide.	*job_ID*
OR bar |	You must enter one of the items separated by the bar. You cannot enter more than one item, Do not enter the bar.	[-h | -V]
Parenthesis ( )	Must be entered exactly as shown	-X "*exception_cond*([*params*])::*acti on*] ...
Option or variable in square brackets [ ]	The argument within the brackets is optional. Do not enter the brackets.	lsid [-h]
Shell prompts	◆ C shell: `%` ◆ Bourne shell and Korn shell: `$` ◆ root account: `#` Unless otherwise noted, the C shell prompt is used in all command examples	**`% cd /bin`**

What’s New in the Platform LSF Version 6.0

Platform LSF Version 6.0 introduces the following new features:

Policy management

Goal-oriented SLA-driven scheduling

Goal-oriented SLA-driven 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.
- They define a “just-in-time” service-level agreement between LSF administrators and LSF users.

You implement your SLA scheduling policies in *service classes* associated with your projects and users. Each service class defines how many jobs should be run to meet different 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.

You use the `bsla` command to track the progress of your projects and see whether they are meeting the goals of your policy.

See Chapter 15, "Goal-Oriented SLA-Driven Scheduling" for more information.

Platform LSF License Scheduler

Platform LSF License Scheduler ensures that higher priority work never has to wait for a license. Prioritized sharing of application licenses allows you to make policies that control the way software licenses are shared among different users in your organization.

You configure your software license distribution policy and LSF intelligently allocates licenses to improve quality of service to your end users while increasing throughput of high-priority work and reducing license costs.

It has the following features:

- Applies license distribution policies fairly among multiple projects cluster-wide
- Easily configurable distribution policies; instead of assigning equal share of licenses to everyone, you can give more licenses to larger or more important projects
- Guaranteed access to a minimum portion of licenses, no matter how heavily loaded the system is
- Controls the distribution of licenses among jobs and tasks it manages and still allows users to check out licenses directly
- Preempts lower priority jobs and releases their licenses to allow higher priority jobs to get the license and run.
- Provides visibility of license usage with `blusers` command

See *Using Platform LSF License Scheduler* for installation and configuration instructions.

Platform LSF license-aware scheduling is available as separately installable add-on packages located in `/license_scheduler/` on the Platform FTP site (`ftp.platform.com/`).

Job-level exception management

Configure hosts and queues so that LSF takes appropriate action automatically when it detects exceptional conditions while jobs are running. Customize what exceptions are detected, and their corresponding actions.

LSF detects:

- Job exceptions:
  - Job underrun—job ends 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)
  - Idle job—running job consumes less CPU time than expected (in terms of cputime/runtime)

- Host exceptions:
  - LSF detects "black hole" or "job-eating" hosts. LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure.
  - A host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such problem hosts exit abnormally.

See Chapter 4, "Working with Hosts" for more information.

Queue-based fairshare

Prevents starvation of low-priority work and ensures high-priority jobs get the resources they require by sharing resources among queues. Queue-based fairshare extends your existing user- and project-based fairshare policies by enabling flexible slot allocation per queue based on slot share units you configure.

See Chapter 14, "Fairshare Scheduling" for more information.

User fairshare by queue priority

Improves control of user-based fairshare by taking queue priority into account for dispatching jobs from different queues against the same user fairshare policy. Within the queue, dispatch order is based on share quota.

See Chapter 14, "Fairshare Scheduling" for more information.

Job group support

Use LSF job groups to organize and control a collection of individual jobs in higher level work units 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, you can organize jobs around groups that are meaningful to your business: 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.

Jobs groups increase end-user productivity by reducing complexity:

- Submit, view, and control jobs according to their groups rather than looking at individual jobs
- Create job group hierarchies
- Move jobs in and out of job groups as needed
- Kill, stop resume and send job control actions to entire job groups
- View job status by group

See Chapter 6, "Managing Jobs" for more information.

High Performance Computing

Dynamic ptile enforcement

Parallel jobs now have a flexible choice of the number of CPUs in the different kinds of hosts in a heterogeneous cluster.

Improves the performance and throughput of parallel jobs by setting multiple `ptile` values in a `span` string according to the CPU configuration of the host type or model.

You can specify various `ptile` values in the queue (RES_REQ in `lsb.queues`, or at job submission with `bsub -R`):

- Default `ptile` value, specified by *n* processors. For example:

  ```
  span[ptile=4]
  ```

  LSF allocates 4 processors on each available host, regardless of how many processors the host has.

- Predefined `ptile` value, specified by '!'. For example:

  ```
  span[ptile='!']
  ```

  LSF uses the predefined maximum job slot limit in `lsb.hosts` (MXJ per host type/model) as its value.

- Predefined `ptile` value with optional multiple `ptile` values, per host type or host model. 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.

See Chapter 13, "Specifying Resource Requirements" for more information.

Resource requirement specification for advance reservation

You no longer need to specify a host list manually for your advance reservations. Specify a resource requirement string with the `-R` option of `brsvadd` instead of or in addition to a list of hosts. This makes advance reservation specification more flexible by reserving host slots based on your specific resource requirements. Only hosts that satisfy the resource requirement expression are reserved.

See Chapter 35, "Advance Reservation" for more information.

Administration and diagnosis

Scheduler dynamic debug

Enables dynamic debugging of the LSF scheduler daemon (`mbschd`) without reconfiguring the cluster. Administrators no longer need to run `badmin mbdrestart` to debug the LSF scheduler:

```
badmin schddebug [-c class_name] [-l debug_level] [-f
logfile_name] [-o]
```

```
badmin schdtime [-l timing_level] [-f logfile_name] [-o]
```

See Chapter 42, "Troubleshooting and Error Messages" for more information.

Administrator action messages

Improves communication of LSF status to users. Users know the reason for the administrator actions, and administrators can easily communicate actions to users.

Administrators can attach a message to `mbatchd` restart, and host and queue operations:

- Use the `-C` option of `badmin mbdrestart` to log an administrator comment in `lsb.events`. For example,

  ```
  % badmin mbdrestart -C "Configuration change"
  ```

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

- Use the `-C` option of `badmin hclose` and `badmin hopen` to log an administrator comment in `lsb.events`. For example,

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

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

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

To see administrator comments, users run `badmin hist`, `badmin mbdhist`, `badmin hhist`, or `badmin qhist`.

See Chapter 3, "Working with Your Cluster", Chapter 4, "Working with Hosts", and Chapter 5, "Working with Queues" for more information.

Platform LSF Reports

Understand cluster operations better, so that you can improve performance and troubleshoot configuration problems.

Platform LSF Reports provides a lightweight reporting package for single LSF clusters. It provides simple two-week reporting for smaller LSF clusters (about 100 hosts, 1,000 jobs/day) and shows trends for basic cluster metrics by user, project, host, resource and queue.

LSF Reports provides the following historical information about a cluster:

- Cluster load

  Trends the LSF internal load indices: status, r15s, r1m, r15m, ut, pg, ls, it, swp, mem, tmp, and io.
- Cluster service level

  Shows the average cluster service level using the following metrics: CPU time, memory and swap consumption, job runtime, job pending time, and job turnaround time
- Cluster throughput

  Shows the amount of work pushed through the cluster, using both accounting information (total number of submitted, completed, and exited jobs) and sampled information (the minimum, maximum, and average number of running and pending jobs, by state and type).
- Shared resource usage

  Shows the total, free, and used shared resources for the cluster.
- Reserved resource usage

  Shows the actual usage of reserved resources.
- License usage

  Shows peak, average, minimum, and maximum license usage by feature.
- License consumption

  Shows license minutes consumed by user, feature, vendor, and server.

See *Platform LSF Reports Reference* for installation and configuration instructions.

Platform LSF Reports is available as separately installable add-on packages located in `/lsf_reports/` on the Platform FTP site (`ftp.platform.com/`).

Run-time enhancements

Thread limit enforcement

Control job thread limit like other limits. Use `bsub -T` to set the limit of the number of concurrent threads for the whole job. The default is no limit. In the queue, set THREADLIMIT to limit the number of concurrent threads that can be part of a job. Exceeding the limit causes the job to terminate.

See Chapter 26, "Runtime Resource Usage Limits" for more information.

Non-normalized job run time limit

Presents consistent job run time limits no matter which host runs the job. With non-normalized job run limit configured, job run time is not normalized by CPU factor.

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 Chapter 26, "Runtime Resource Usage Limits" for more information.

Resource allocation limit display (blimits command)

Improves visibility to resource allocation limits. If your job is pending because some configured resource allocation limit has been reached, you can find out what limits may be blocking your job.

Use the `blimits` command to show the dynamic counters of each resource allocation limit configured in `lsb.resources`.

See Chapter 16, "Resource Allocation Limits" for more information.

Upgrade and Compatibility Notes

UPGRADE document

To upgrade to LSF Version 6.0, follow the steps in `upgrade.html`.

API Compatibility between LSF 5.x and Version 6.0

Full backward compatibility: your applications will run under LSF Version 6.0 without changing any code.

The Platform LSF Version 6.0 API is fully compatible with the LSF Version 5.*x* and Version 4.*x* API. An application linked with the LSF Version 5.*x* and Version 4.*x* library will run under LSF Version 6.0 without relinking.

To take full advantage of new Platform LSF Version 6.0 features, you should recompile your existing LSF applications with LSF Version 6.0.

Server host compatibility Platform LSF

You must upgrade the LSF master hosts in your cluster to Version 6.0.

LSF 5.*x* servers are compatible with Version 6.0 master hosts. All LSF 5.x features are supported by 6.0 master hosts except:

To use new features introduced in Platform LSF Version 6.0, you must upgrade all hosts in your cluster to 6.0.

Platform LSF MultiCluster

You must upgrade the LSF master hosts in all clusters to Version 6.0.

New configuration parameters and environment variables

The following new parameters and environment variables have been added for LSF Version 6.0:

lsb.hosts EXIT_RATE specifies a threshold in minutes for exited jobs

lsb.params

- EADMIN_TRIGGER_DURATION defines how often `LSF_SERVERDIR/eadmin` is invoked once a job exception is detected.
- JOB_EXIT_RATE_DURATION defines how long LSF waits before checking the job exit rate for a host.
- ABS_RUNLIMIT—if set, the run time limit specified by the `-W` option of `bsub`, or the RUNLIMIT queue parameter in `lsb.queues` is not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit.

lsb.queues

- DISPATCH_ORDER defines an ordered cross-queue fairshare set
- JOB_IDLE specifies a threshold for idle job exception handling
- JOB_OVERRUN specifies a threshold for job overrun exception handling
- JOB_UNDERRUN specifies a threshold for job underrun exception handling
- RES_REQ accepts multiple ptile specifications in the span section for dyamic ptile enforcement

- SLOT_POOL is the name of the pool of job slots the queue belongs to for queue-based fairshare
- SLOT_SHARE specifies the share of job slots for queue-based fairshare, representing the percentage of running jobs (job slots) in use from the queue
- THREADLIMIT limits the number of concurrent threads that can be part of a job. Exceeding the limit causes the job to terminate
- RUNLIMIT—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 to a queue with a run limit configured.

Environment variables

- LSB_SUB_EXTSCHED_PARAM

  Value of external scheduling options specified by `bsub -extsched`, or queue-level MANDATORY_EXTSCHED or DEFAULT_EXTSCHED
- LSB_SUB_JOB_WARNING_ACTION

  Value of job warning action specified by `bsub -wa`
- LSB_SUB_JOB_WARNING_TIME_PERIOD

  Value of job warning time period specified by `bsub -wt`

New command options and output

The following command options and output have changed for LSF Version 6.0:

bacct

- `-sla` *service_class_name* displays accounting statistics for jobs that ran under the specified service class
- `-x` displays jobs that have triggered a job exception (overrun, underrun, idle)

badmin

- `schddebug` sets message log level for `mbschd` to include additional information in log files
- `schdtime` sets timing level for `mbschd` to include additional timing information in log files
- `-C` *comment* logs the text of *comment* as an administrator comment record to `lsb.events` for the following subcommands:
  - `mbdrestart`
  - `qopen`
  - `qclose`
  - `qact`
  - `qinact`
  - `hopen`
  - `hclose`

bhist

`-l` displays:

- Job group modification
- Configured thread limit

bhosts

- `-x` displays hosts whose job exit rate has exceeded the threshold configured by EXIT_RATE in `lsb.hosts` for longer than JOB_EXIT_RATE_DURATION configured in `lsb.params`, and are still high

- `-l` displays the comment text if the LSF administrator specified an administrator comment with the `-C` option of the `badmin` host control commands `hclose` or hopen

bjobs

- `-g` *job_group_name* displays information about jobs attached to the specified job group
- `-l` displays the thread limit for the job
- `-sla` *service_class_name* displays jobs belonging to the specified service class
- `-x` displays unfinished jobs that have triggered a job exception (overrun, underrun, idle)

bkill

- `-g` *job_group_name* operates only on jobs in the specified job group
- `-sla` *service_class_name* operates on jobs belonging to the specified service class.

bmod

- `-g` *job_group_name* | `-gn`
- `-sla` *service_class_name* | `-slan`

bqueues

`-l` displays:

- Configured job exception thresholds and number of jobs in each exception state for the queue
- The job slot share (SLOT_SHARE) and the name of the share pool (SLOT_POOL) that the queue belongs to for queue-based fairshare
- DISPATCH_ORDER in a master queue for cross-queue fairshare
- The comment text if the LSF administrator specified an administrator comment with the `-C` option of the queue control commands `qclose`, `qopen`, `qact`, and `qinact`, `qhist`

bresume

`-g` *job_group_name* resumes only jobs in the specified job group

brsvadd

`-R` selects hosts for the reservation according to the specified resource requirements

bstop

- `-g` *job_group_name* suspends only jobs in the specified job group
- `-sla` *service_class_name* suspends jobs belonging to the specified service class

bsub

- `-g` *job_group_name* submits jobs in the specified job group
- `-R` accepts multiple ptile specifications in the span section for dyamic ptile enforcement
- `-sla` *service_class_name* specifies the service class where the job is to run
- `-T` *thread_limit* sets the limit of the number of concurrent threads to thread_limit for the whole job.
- `-W`—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.

New files added to installation

The following new files have been added to the Platform LSF Version 6.0 installation:

- `LSB_CONFDIR/`*cluster_name*`/configdir/lsb.serviceclasses`
- `LSF_BINDIR/bgadd`
- `LSF_BINDIR/bgdel`
- `LSF_BINDIR/bjgroup`
- `LSF_BINDIR/blimits`
- `LSF_BINDIR/bsla`
- `LSF_SERVERDIR/eadmin`
- `LSF_LIBDIR/schmod_jobweight.so`

Symbolic links to LSF files

If your installation uses symbolic links to other files in these directories, you must manually create links to these new files.

New accounting and job event fields

The following fields have been added to `lsb.acct` and `lsb.events`:

lsb.acct

- JOB_FINISH:

  sla (%s) is the SLA service class name under which the job runs.

lsb.events

- JOB_NEW:
  - sla (%s) is the SLA service class name under which the job runs
  - SLArunLimit (%d) is the absolute run time limit of the job for SLA service classes
  - jobGroup (%s) is the job group under which the job runs
- JOB_MODIFY2:
  - sla (%s) is the SLA service class name that the job is to be attached to
  - jobGroup (%s) is the job group under which the job runs
- JOB_EXECUTE:

  SLAscaledRunLimit (%d) is the run time limit for the job scaled by the execution host
- QUEUE_CTRL:

  ctrlComments (%s) is the administrator comment text from the `-C` option of `badmin` queue control commands `qclose`, `qopen`, `qact`, and `qinact`
- HOST_CTRL:

  ctrlComments (%s) is the administrator comment text from the `-C` option of `badmin` host control commands `hclose` and `hopen`
- MBD_DIE:

  ctrlComments (%s) is the administrator comment text from the `-C` option of `badmin mbdrestart`

Learning About Platform Products

World Wide Web and FTP

The latest information about all supported releases of Platform LSF is available on the Platform Web site at `www.platform.com`. Look in the Online Support area for current README files, Release Notes, Upgrade Notices, Frequently Asked Questions (FAQs), Troubleshooting, and other helpful information.

The Platform FTP site (`ftp.platform.com`) also provides current README files, Release Notes, and Upgrade information for all supported releases of Platform LSF.

Visit the Platform User Forum at `www.platformusers.net` to discuss workload management and strategies pertaining to distributed and Grid Computing.

If you have problems accessing the Platform web site or the Platform FTP site, contact `support@platform.com`.

Platform training

Platform's Professional Services training courses can help you gain the skills necessary to effectively install, configure and manage your Platform products. Courses are available for both new and experienced users and administrators at our corporate headquarters and Platform locations worldwide.

Customized on-site course delivery is also available.

Find out more about Platform Training at `www.platform.com/training`, or contact `Training@platform.com` for details.

README files and release notes and UPGRADE

Before installing LSF, be sure to read the files named `readme.html` and `release_notes.html`. To upgrade to Version 6.0, follow the steps in `upgrade.html`.

You can also view these files from the Download area of the Platform Online Support Web page.

Platform documentation

Documentation for Platform products is available in HTML and PDF format on the Platform Web site at `www.platform.com/services/support/docs_home.asp`.

Technical Support

Contact Platform Computing or your LSF vendor for technical support.

Email support@platform.com

World Wide Web www.platform.com

Phone

- North America: +1 905 948 4297
- Europe: +44 1256 370 530
- Asia: +86 10 6238 1125

Toll-free phoneó 1-877-444-4LSF (+1 877 444 4573)

Mail Platform Support
Platform Computing Corporation
3760 14th Avenue
Markham, Ontario
Canada L3R 3T7

When contacting Platform, please include the full name of your company.

We'd like to hear from you

If you find an error in any Platform documentation, or you have a suggestion for improving it, please let us know:

Email doc@platform.com

Mail Information Development
Platform Computing Corporation
3760 14th Avenue
Markham, Ontario
Canada L3R 3T7

Be sure to tell us:

- The title of the manual you are commenting on
- The version of the product you are using
- The format of the manual (HTML or PDF)

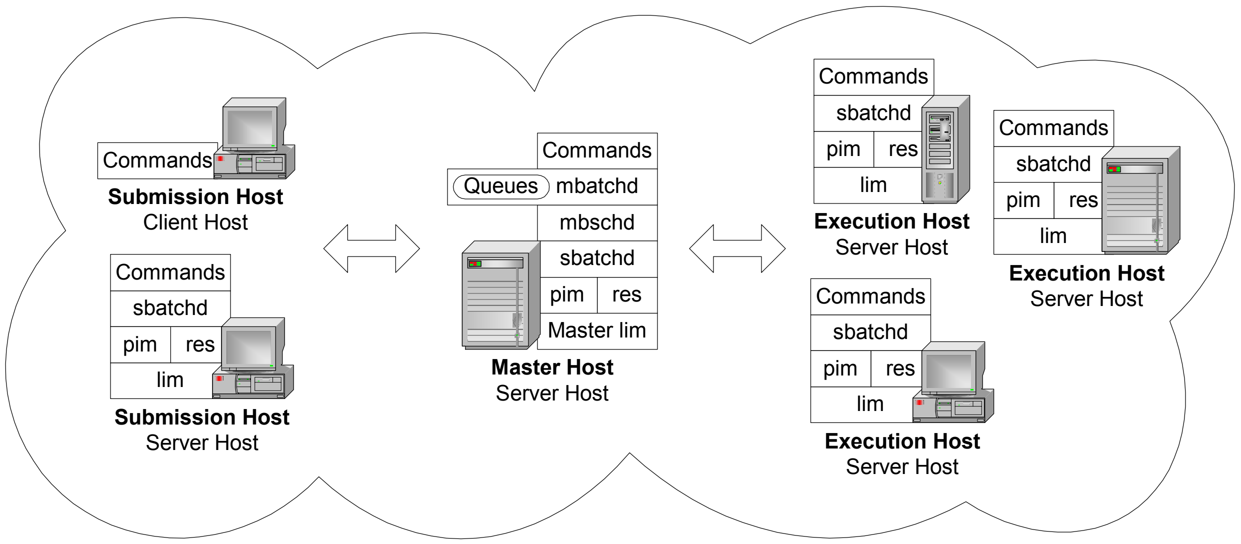
C H A P T E R

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

mbatchd	**job requests and dispatch**
mbschd	**job scheduling**
sbatchd	**job execution**
res	
lim	**host information**
pim	**job process information**

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 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 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 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

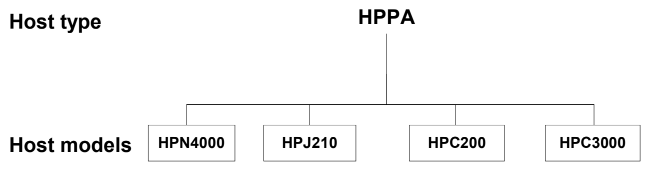
Configuration

- `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.bhosts`—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.

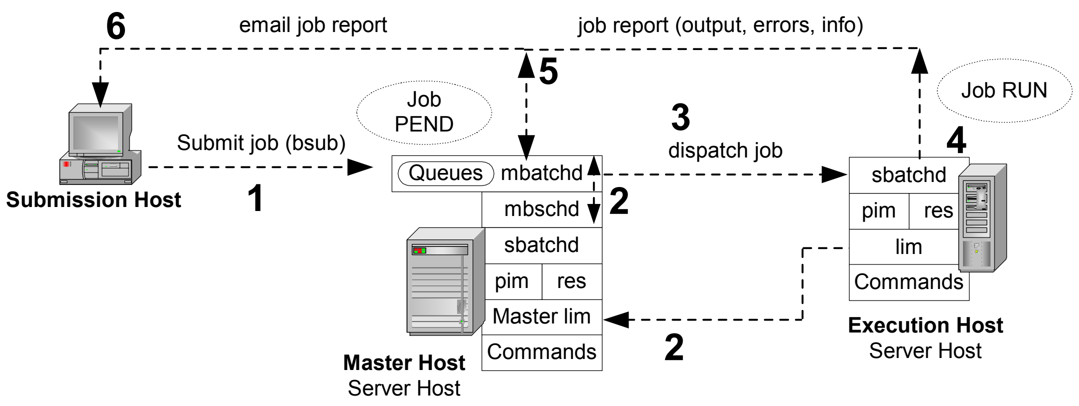
Commands

- `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

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

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. 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.

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.

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 117 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.

See “Default Directory Structures” on page 58 for more information on LSF installation directories.

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 master host selection is based on the order in which hosts are listed in the `lsf.cluster.`*`cluster_name`* file. If the first host in the file is available, that host acts as the master. If the first host is unavailable, the second host takes over, and so on. 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 state 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 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.

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 96 and “Handling Job Exceptions” on page 109 for more information about job-level exception management.

P A R T

I

Managing Your Cluster

Contents

C H A P T E R

3

Working with Your Cluster

Contents

Viewing Cluster Information

LSF provides commands for users to get 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	`lsid`
Cluster name	`lsid`
Current master host	`lsid`
Cluster administrators	`lsclusters`
Configuration parameters	`bparams`

Viewing LSF version, cluster name, and current master host

Use the `lsid` command to display the version of LSF, the name of your cluster, and the current master host:

```
% lsid
Platform LSF 6.0, Oct 31 2003
Copyright 1992-2004 Platform Computing Corporation

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

Viewing cluster administrators

Use the `lsclusters` command to find out who your cluster administrator is and see a summary of your cluster:

```
% lsclusters
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 `lsclusters`.

Viewing configuration parameters

Use the `bparams` command to display the generic configuration parameters of LSF. These include default queues, default host or host model for CPU speed scaling, job dispatch interval, job checking interval, job accepting interval, etc.

```
% bparams
Default Queues:  normal idle
Default Host Specification:  DECAXP
Job Dispatch Interval:  20 seconds
Job Checking Interval:  15 seconds
Job Accepting Interval:  20 seconds
```

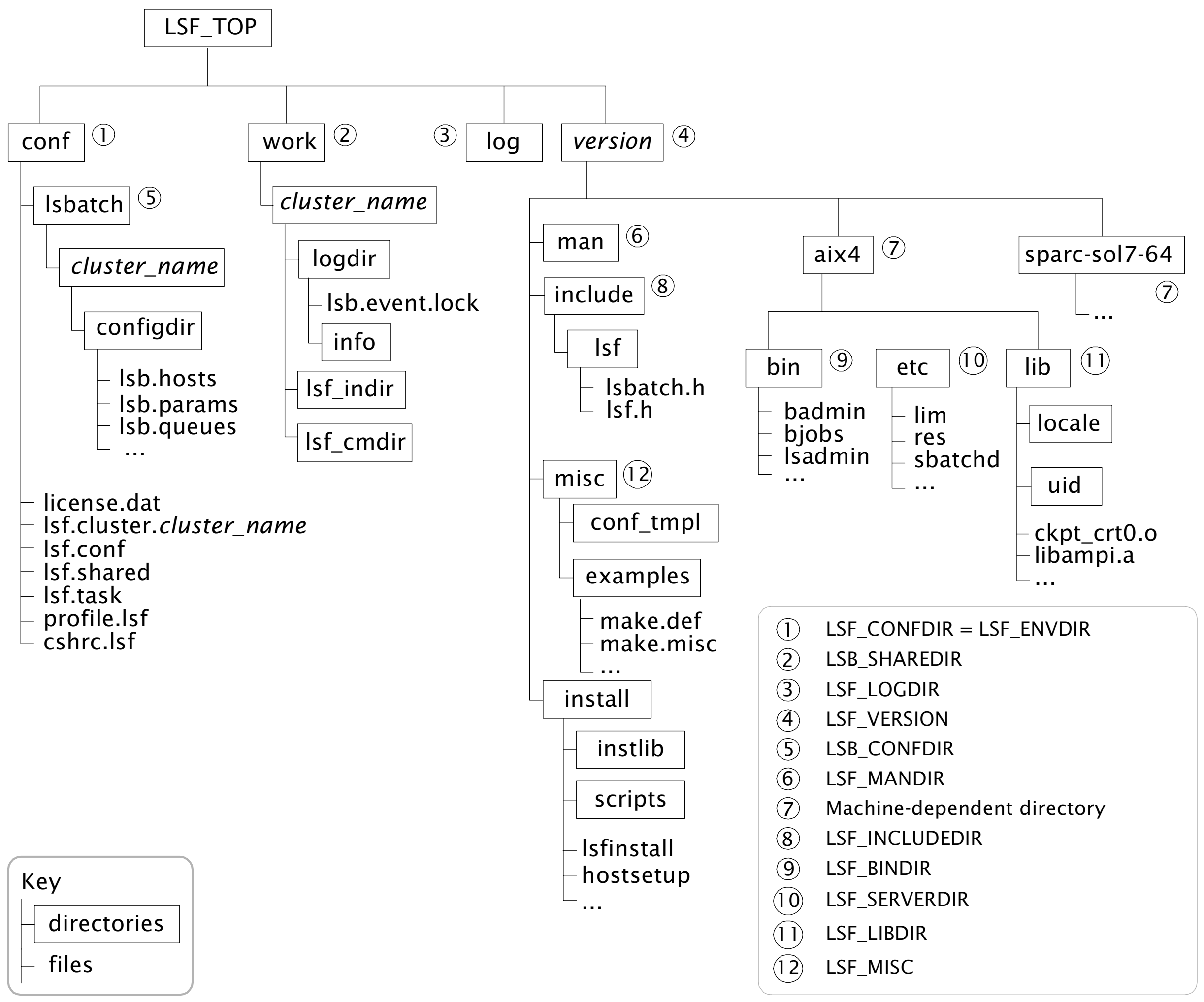
Use the `-l` option of `bparams` to display the information in long format, which gives a brief description of each parameter as well as the name of the parameter as it appears in `lsb.params`.

```
% 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
```

Default Directory Structures

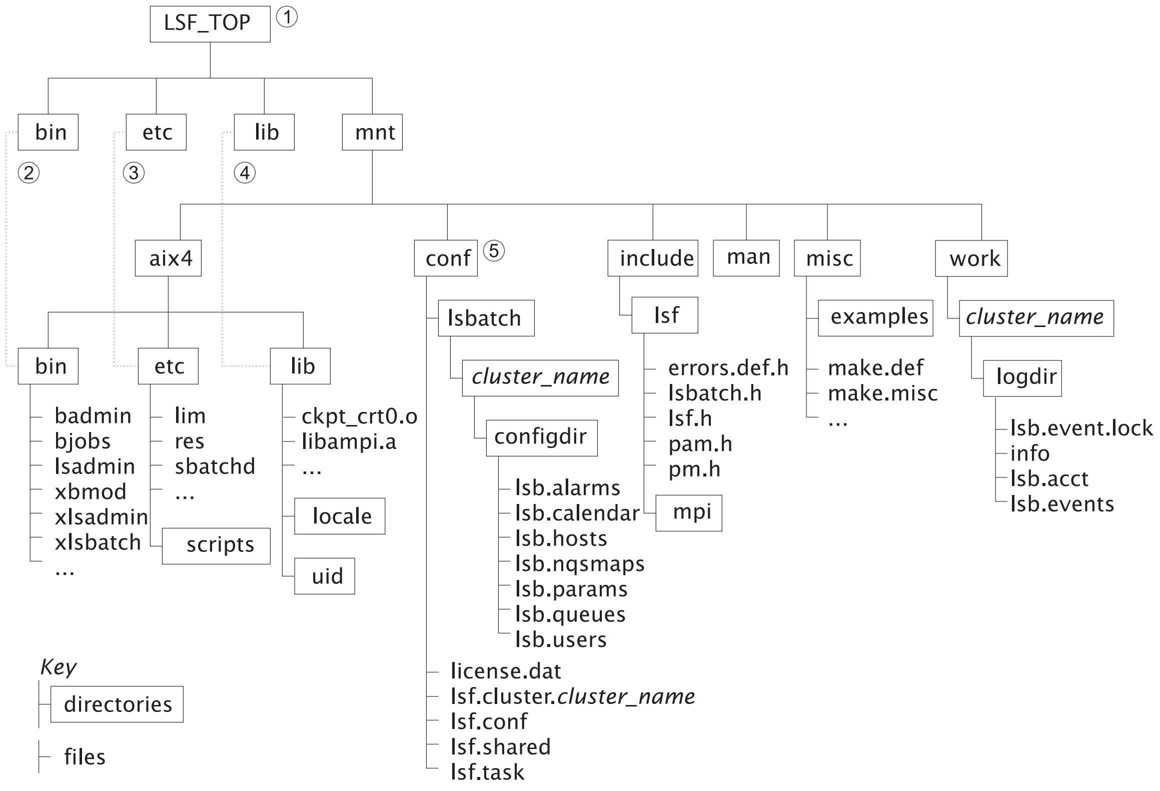
UNIX

The following diagram shows a typical directory structure for a new UNIX installation. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Pre-4.2 UNIX installation directory structure

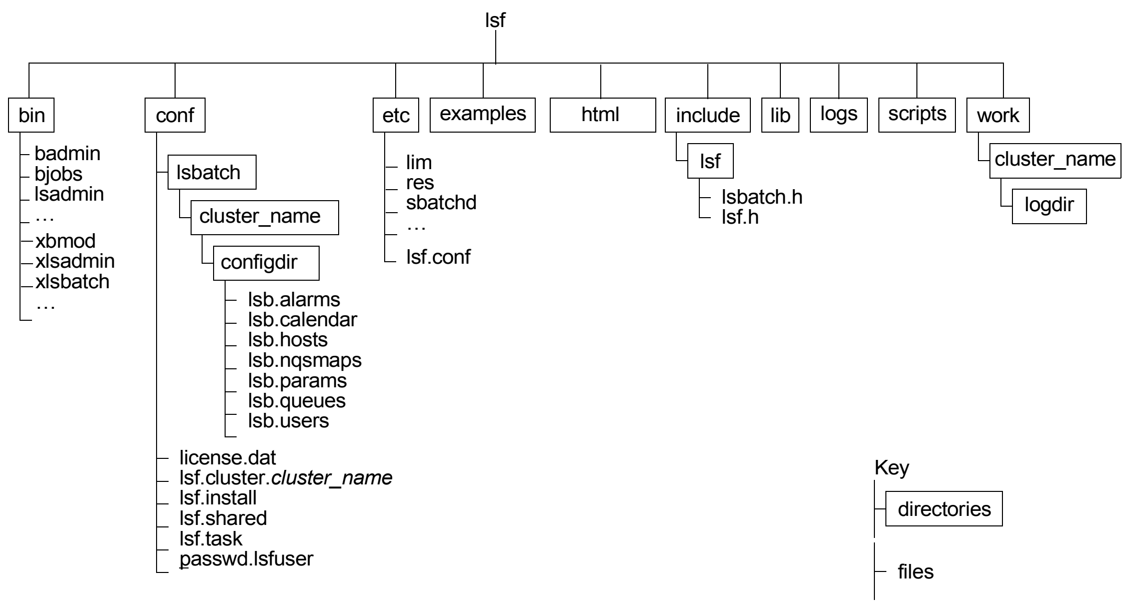
The following diagram shows a cluster installed with `lsfsetup`. It uses the pre-4.2 directory structure.



① By default `/usr/local/lsf`
② LSF_BINDIR, symbolic link to `LSF_TOP/mnt/aix4/bin`
③ LSF_SERVERDIR, symbolic link to `LSF_TOP/mnt/aix4/etc`
④ LSF_LIBDIR, symbolic link to `LSF_TOP/mnt/aix4/lib`
⑤ LSF_CONFDIR

Windows

The following diagram shows the directory structure for a default 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.

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.

Adding cluster administrators

1. In the `ClusterAdmins` section of `lsf.cluster.`*`cluster_name`*, specify the list of cluster administrators following ADMINISTRATORS, separated by spaces. The first administrator in the list is the primary LSF administrator. All others are cluster administrators. You can specify user names and group names. 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

Prerequisites

To control all daemons in the cluster, you must:

- Be logged on as root or a user listed in the `/etc/lsf.sudoers` file

  See the *Platform LSF 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	lsfstartup	Must be root or a user listed in `lsf.sudoers` for all these commands
	Shut down	lsfshutdown	
`sbatchd`	Start	badmin hstartup [host_name ... | all]	Must be root or a user listed in `lsf.sudoers` for the startup command
	Restart	badmin hrestart [host_name ... | all]	Must be root or the LSF administrator for other commands.
	Shut down	badmin hshutdown [host_name ... | all]	
`mbatchd` `mbschd`	Restart	badmin mbdrestart	Must be root or the LSF administrator for these commands
	Shut down	1 badmin hshutdown 2 badmin mbdrestart	
	Reconfigure	badmin reconfig	
RES	Start	lsadmin resstartup [host_name ... | all]	Must be root or a user listed in `lsf.sudoers` for the startup command
	Shut down	lsadmin resshutdown [host_name ... | all]	Must be the LSF administrator for other commands
	Restart	lsadmin resrestart [host_name ... | all]	
LIM	Start	lsadmin limstartup [host_name ... | all]	Must be root or a user listed in `lsf.sudoers` for the startup command
	Shut down	lsadmin limshutdown [host_name ... | all]	Must be the LSF administrator for other commands
	Restart	lsadmin limrestart [host_name ... | all]	
	Restartall in cluster	lsadmin reconfig	

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 on the current master host is 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

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

If you add a host to a host group, or a host to a queue, the new host is not recognized by jobs that were submitted before you reconfigured. If you want the new host to be recognized, you must restart `mbatchd`.

Restarting mbatchd

Run `badmin 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`.

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 `badmin reconfig`.

Logging a comment when restarting mbatchd

Use the `-C` option of `badmin mbdrestart` to log an administrator comment in `lsb.events`. For example,

```
% badmin mbdrestart -C "Configuration change"
```

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

Use `badmin hist` or `badmin mbdhist` to display administrator comments for `mbatchd` restart.

Shutting 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.

Reconfiguring Your Cluster

After changing LSF configuration files, you must tell LSF to reread the files to update the configuration. The commands you can use to reconfigure a cluster are:

- `lsadmin reconfig`
- `badmin reconfig`
- `badmin 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 ...
`hosts`	`badmin reconfig`	reloads configuration files
`lsb.hosts`	`badmin reconfig`	reloads configuration files
`lsb.modules`	`badmin reconfig`	reloads configuration files
`lsb.nqsmaps`	`badmin reconfig`	reloads configuration files
`lsb.params`	`badmin reconfig`	reloads configuration files
`lsb.queues`	`badmin reconfig`	reloads configuration files
`lsb.resources`	`badmin reconfig`	reloads configuration files
`lsb.users`	`badmin reconfig`	reloads configuration files
`lsf.cluster.`*`cluster_name`*	`lsadmin reconfig` AND `badmin mbdrestart`	reconfigures LIM, reloads configuration files, and restarts `mbatchd`
`lsf.conf`	`lsadmin reconfig` AND `badmin mbdrestart`	reconfigures LIM and reloads configuration files, and restarts `mbatchd`
`lsf.shared`	`lsadmin reconfig` AND `badmin mbdrestart`	reconfigures LIM, reloads configuration files, and restarts `mbatchd`
`lsf.sudoers`	`badmin reconfig`	reloads configuration files
`lsf.task`	`lsadmin reconfig` AND `badmin reconfig`	reconfigures LIM and reloads configuration files

Reconfiguring the cluster with lsadmin and badmin

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

```
% lsadmin reconfig
Checking configuration files ...
No errors found.

Do you really want to restart LIMs on all hosts? [y/n] y
Restart LIM on <hosta> ...... done
Restart LIM on <hostc> ...... done
Restart LIM on <hostd> ...... done
```

The `lsadmin reconfig` command checks for configuration errors.

If no errors are found, you are asked to confirm that you want to restart `lim` on all hosts and `lim` is reconfigured. If fatal errors are found, reconfiguration is aborted.

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

```
% badmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The `badmin reconfig` command checks for configuration errors.

If no fatal errors are found, you are asked to confirm reconfiguration. If fatal errors are found, reconfiguration is aborted.

Reconfiguring the cluster by restarting mbatchd

Run `badmin mbdrestart` to restart `mbatchd`:

```
% badmin mbdrestart
Checking configuration files ...
No errors found.
Do you want to restart? [y/n] y
MBD restart initiated
```

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.

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.

Viewing configuration errors

You can view configuration errors by using the following commands:

- `lsadmin ckconfig -v`
- `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 will not reconfigure the system because the system would lose all its licenses and stop working.

If you have multiple license servers, reconfiguration will proceed as long as LSF can contact at least one license server. In this case, LSF will still lose the licenses on servers that are down, so LSF may have fewer licenses available after reconfiguration.

C H A P T E R

4

Working with Hosts

Contents

Host States

Host states describe 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 states.

bhosts

Displays the current state of the host:

STATUS	Description
`ok`	Host is available to accept and run new batch jobs.
`unavail`	Host is down, or LIM and `sbatchd` are unreachable.
`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 will not accept new batch jobs:

STATUS	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 lsadmin limlock. Running jobs are suspended (SSUSP).
`closed_Wind`	Host is closed by a dispatch window defined in lsb.hosts. Running jobs are not affected.

```
% 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    -     -     -     -       -     -    -     -     -      -      -
```

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., lsrun). The value of a load index exceeded a threshold (configured in lsf.cluster.*cluster_name*, displayed by lshosts -l). Indices that exceed thresholds are identified with an asterisk (*).
`lockW`	Does not affect batch jobs, only used for remote task placement (i.e., lsrun). Host is locked by a run window (configured in lsf.cluster.*cluster_name*, displayed by lshosts -l).
`lockU`	Will not accept new batch jobs or remote tasks. An LSF administrator or root explicitly locked the host (i.e., lsadmin limlock) or an exclusive batch job (i.e., bsub -x) is running on the host. Running jobs are not affected.
`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
...
```

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	`bhosts`
Detailed server host information	`bhosts -l` and `lshosts -l`
Host load by host	`lsload`
Host architecture information	`lshosts`
Host history	`badmin hhist`
Host model and type information	`lsinfo`
Viewing job exit rate and load for hosts	`bhosts -l` and `bhosts -x`

Viewing all hosts in the cluster and their status

Run `bhosts` to display information about all hosts and their status. For example:

```
% bhosts
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
```

Viewing detailed server host information

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. For example:

```
% bhosts -l hostB
HOST  hostB
STATUS   CPUF   JL/U   MAX   NJOBS   RUN   SSUSP   USUSP   RSV   DISPATCH_WINDO
WS
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
LOAD THRESHOLD USED FOR SCHEDULING:
            r15s  r1m   r15m  ut    pg   io  ls   it   tmp  swp  mem
loadSched   -     -     -     -     -     -   -    -     -    -    -
loadStop    -     -     -     -     -     -   -    -     -    -    -
```

```
% lshosts -l hostB
HOST_NAME:  hostB
type    model    cpuf    ncpus    ndisks    maxmem    maxswp    maxtmp    rexpri   serve
r
Sun4   Ultra2    20.2        2         1      256M      710M      466M         0      Ye
s

RESOURCES: Not defined
RUN_WINDOWS:  (always open)

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster LSF_Make LSF_Parallel)

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

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

Run `lsload` to see load levels for each host. For example:

```
% 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 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)
hostB      ALPHA  DEC3000 10.0     1    94M   168M    Yes  (alpha 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 532 for instructions on measures to take.

When automatic detection of host type or model fails, the type or model is set to `DEFAULT`. LSF will work on the host. A `DEFAULT` model may be inefficient because of incorrect CPU factors. A `DEFAULT` type may cause binary incompatibility because a job from a `DEFAULT` host type can be migrated to another `DEFAULT` host type.

Viewing host history

Run `badmin hhist` to view the history of a host such as when it is opened or closed. For example:

```
% badmin 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>.
```

Viewing host model and type information

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
```

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
```

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

```
% lim -t
Host Type                : SOL732
Host Architecture        : SUNWUltra2_200_sparcv9
Matched Type             : SOL732
Matched Architecture     : SUNWUltra2_300_sparc
Matched Model            : Ultra2
CPU Factor               : 20.2
```

Viewing job exit rate and load for hosts

Use `bhosts` to display the exception threshold for job exit rate and the current load value for hosts. For 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    -     -     -     -       -     -    -     -     -      -      -

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

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 will be 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
```

Controlling Hosts

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

Closing a host

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.

Opening a host

Run `badmin hopen`:

```
% badmin hopen hostB
Open <hostB> ...... done
```

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 on or more time windows in the DISPATCH_WINDOW column. For example:

```
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 `badmin reconfig` to reconfigure `mbatchd`.
4. Run `bhosts -l` to display the dispatch windows.

Logging a comment when closing or opening a host

Use the `-C` option of `badmin hclose` and `badmin hopen` to log an administrator comment in `lsb.events`. For example,

```
% badmin 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:

```
% badmin hclose -C "backup" hostA
```

followed by

```
% badmin hclose -C "Weekly backup" hostA
```

will generate records in `lsb.events`:

```
"HOST_CTRL" "6.0 1050082346 1 "hostA" 32185 "lsfadmin" "backup"
"HOST_CTRL" "6.0 1050082373 1 "hostA" 32185 "lsfadmin" "Weekly backup"
```

Use `badmin hist` or `badmin hhist` to display administrator comments for closing and opening hosts. For example:

```
% 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    -     -     -     -       -     -    -     -     -      -      -

 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

Use `lsfinstall` to add a host to an LSF cluster.

Contents

See the *Platform LSF Reference* for more information about `lsfinstall`.

Adding a host of an existing type using lsfinstall

Compatibility notice

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/6.0 directory. The default is `/usr/share/lsf/`6.0. If the host type currently exists, there will be a subdirectory with the name of the host type. If it does not exist, go to "Adding a host of a new type using lsfinstall" on page 77.
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:
      i. The name of the host.
      ii. The model and type, or specify ! to automatically detect the type or model.
      iii. Specify **`1`** for LSF server or **`0`** for LSF client. For example:

```
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/6.0/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 532 to fix the problem.
- If any host type or host model is DEFAULT, follow the steps in "DEFAULT host type or model" on page 532 to fix the problem.

Adding a host of a new type using lsfinstall

Compatibility Notice **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/6.0 directory. The default is `/usr/share/lsf/6.0`. 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 "Adding a host of an existing type using lsfinstall" on page 76.
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/6.0/install`
5. Edit `install.config`:
   a. For LSF_TARDIR, specify the path to the tar file. For example:
      ```
      LSF_TARDIR="/usr/share/lsf_distrib/6.0"
      ```
   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/6.0/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 532 to fix the problem.
    - If any host type or host model is DEFAULT, follow the steps in "DEFAULT host type or model" on page 532 to fix the problem.

Removing 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 and Removing Hosts Dynamically

By default, all configuration changes made to LSF are static. You must manually change the configuration and restart the cluster (or at least all master candidates). Dynamic host configuration allows you to add hosts to the cluster or remove them without manually changing the configuration.

WARNING **When the dynamic host configuration is enabled, any host will be able to join the cluster. You can limit which hosts can be LSF hosts with the parameter LSF_HOST_ADDR_RANGE in `lsf.cluster.`*`cluster_name`*.**

How dynamic host configuration works

Master LIM For dynamic host configuration, the master LIM:

- Receives request to add hosts
- Informs other master candidates to refresh themselves when a host is added or removed
- Detects host unavailability and, if LSF_DYNAMIC_HOST_TIMEOUT is defined, removes unavailable hosts that are not master candidates

Master candidate LIMs (LSF_MASTER_LIST)

To enable dynamic host configuration, you must define LSF_MASTER_LIST in `lsf.conf`. Specify a list of hosts that are candidates to become the master host for the cluster.

This set of hosts reads the LSF configuration files when a new host is added to the cluster; other hosts (*slave hosts*) only receive the host configuration from master LIM. LSF_MASTER_LIST also identifies the hosts that need to be reconfigured after configuration change.

Master candidate hosts are informed when a new host is added. When a master candidate becomes master host, its LIM receives requests from dynamic hosts to add them to the cluster.

Master candidate hosts should share LSF configuration and binaries.

Slave LIMs Dynamically added LSF hosts that will not be master candidates are *slave hosts*. Each dynamic slave host has its own LSF binaries and local `lsf.conf` and shell environment scripts (`cshrc.lsf` and `profile.lsf`). You must install LSF on each slave host.

If LSF_STRICT_CHECKING is defined in lsf.conf to protect your cluster in untrusted environments, and your cluster has slave hosts that are dynamically added, LSF_STRICT_CHECKING must be configured in the local `lsf.conf` on all slave hosts.

Slave LIMs report their availability to the master LIM when they start. When each slave host starts, it first contacts the master LIM to add itself to the cluster. The master host adds the host if it is not in its host table, or returns `ok` if the host has already been added.

Local resources for slave hosts

Use LSF_LOCAL_RESOURCES in a localized `lsf.conf` to define instances of local resources residing on the slave host:

- For numeric resources, defined name-value pairs:

  `[resourcemap` *`value`*`*`*`resource_name`***`]`**
- For Boolean resources, the value will be the resource name in the form:

  `[resource` *`resource_name`***`]`**

When the slave host calls the master host to add itself, it also reports its local resources. The local resources to be added must be defined in `lsf.shared`.

If the same resource is already defined in `lsf.cluster.`*`cluster_name`* as `default` or `all`, it cannot be added as a local resource. The shared resource overrides the local one.

Resources must already be mapped to hosts in the ResourceMap section of lsf.cluster.cluster_name. If the ResourceMap section does not exist, local resources are not added.

LSF_LOCAL_RESOURCES is usually set in the `slave.config` file during installation. If LSF_LOCAL_RESOURCES are already defined in a local `lsf.conf` on the slave host, `lsfinstall` does not add resources you define in LSF_LOCAL_RESOURCES in `slave.config`. You should not have duplicate LSF_LOCAL_RESOURCES entries in lsf.conf. If local resources are defined more than once, only the last definition is valid.

lsadmin command

The `lsadmin` command is now able to run on a non-LSF host. Use `lsadmin limstartup` to start LIM on a newly added dynamic host.

Master failover

If the master LIM dies, the next master candidate will have the same knowledge as the master LIM about dynamically added hosts in the cluster.

mbatchd

`mbatchd` gets host information from master LIM; when it detects that a host has been added or removed dynamically, `mbatchd` automatically reconfigures itself.

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

Host configuration in lsb.hosts and lsb.queues

For host configuration in `lsb.hosts` and `lsb.queues` to apply to dynamically added hosts, use `default` or `all`, as appropriate, to enable configuration to apply to all hosts in the cluster.

Adding dynamic hosts in a shared file system

If the new dynamically added hosts share the same set of configuration and binary files with normal hosts, you only need to start the LSF daemons on that host and the host is recognized by the master as an LSF host.

New installation

1. Specify the installation options in `install.config`.

   The following parameters are required:

   - LSF_TOP="/*path*"
   - LSF_ADMINS="*user_name* [*user_name* ...]"
   - LSF_CLUSTER_NAME="*cluster_name*"
   - LSF_MASTER_LIST="*host_name* [*host_name* ...]"

     List the hosts that are candidates to become the master host for the cluster.

2. Use `lsfinstall -f install.config` to install LSF.

Existing installation

1. On the master host, configure the following parameters:
   - `lsf.conf`:
     - LSF_MASTER_LIST="*host_name* [*host_name* ...]"

       List the hosts that are candidates to become the master host for the cluster.
     - LSF_DYNAMIC_HOST_TIMEOUT=*timeout*[m | M] (optional)

       Set an optional timeout value in hours or minutes. If the dynamic host is unavailable for longer than the time specified, it is removed from the cluster. To specify a value in minutes, append “`m`” or “`M`” to the timeout value.
   - `lsf.cluster.`*`cluster_name`* (optional)
     - LSF_HOST_ADDR_RANGE=*IP_address* ...
2. Log on as root to each host you want to join a cluster.
3. Use one of the following to set the LSF environment:
   - For `csh` or `tcsh`:
     ```
     % source LSF_TOP/conf/cshrc.lsf
     ```
   - For `sh`, `ksh`, or `bash`:
     ```
     $ . LSF_TOP/conf/profile.lsf
     ```
4. Optionally, run `hostsetup` on each LSF server host.

   You only need to run `hostsetup` if you want LSF to automatically start when the host is rebooted. For example:
   ```
   # cd /usr/share/lsf/5.1/install
   # ./hostsetup --top="/usr/share/lsf" --boot="y"
   ```
   For complete `hostsetup` usage, enter `hostsetup -h`.
5. Use the following commands start LSF:
   ```
   # lsadmin limstartup
   # lsadmin resstartup
   # badmin hstartup
   ```

Adding dynamic hosts in a non-shared file system (slave hosts)

If each dynamic slave host has its own LSF binaries and local `lsf.conf` and shell environment scripts (`cshrc.lsf` and `profile.lsf`), you must install LSF on each slave host.

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

   The following parameters are required:

   - LSF_SERVER_HOSTS="*host_name* [*host_name* ...]"
   - LSF_TARDIR="/*path*"
   - LSF_TOP="/*path*"

   The following parameters are optional:

   - LSF_LIM_PORT=*port_number*

     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.
   - LSF_LOCAL_RESOURCES=*resource* ...

     Defines the local resources for a dynamic host.

     - For numeric resources, defined name-value pairs:

       `[resourcemap` *`value`***`*`***`resource_name`***`]`**
     - For Boolean resources, the value will be the resource name in the form:

       `[resource` *`resource_name`***`]`**

     For example:

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

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

2. Use `lsfinstall -s -f slave.config` to install a dynamic slave host.

   `lsfinstall` creates a local `lsf.conf` for the slave 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* ...
   - LSF_SERVER_HOSTS="*host_name* [*host_name* ...]"
   - LSF_VERSION=6.0

3. Use one of the following to set the LSF environment:

   - For `csh` or `tcsh`:

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

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

4 Optionally, run `hostsetup` on each LSF server host.

   You only need to run `hostsetup` if you want LSF to automatically start when the host is rebooted. For example:

```
# cd /usr/local/lsf/5.1/install
# ./hostsetup --top="/usr/local/lsf" --boot="y"
```

   For complete `hostsetup` usage, enter `hostsetup -h`.

5 Use the following commands start LSF:

```
# lsadmin limstartup
# lsadmin resstartup
# badmin hstartup
```

Limitation The first time a non-shared slave host joins the cluster, daemons on the new host can only be started on local host. For example, the LSF administrator cannot start daemons on `hostB` from `hostA` by using `lsadmin limstartup hostB`. Instead, the first time the host joins the cluster, use:

```
# rsh hostB lsadmin limstartup
```

Allowing only certain hosts to join the cluster

By default, any host can be dynamically added to the cluster. To avoid having unauthorized hosts join the cluster, you can optionally use LSF_HOST_ADDR_RANGE in `lsf.cluster.`*`cluster_name`* to identify a range of IP addresses to identify hosts that are allowed to be dynamically added as LSF hosts.

LSF_HOST_ADDR_RANGE (lsf.cluster.cluster_name)

If a value is defined for LSF_HOST_ADDR_RANGE, security for dynamically adding and removing hosts is enabled, and only hosts with IP addresses within the specified range can be added to or removed from a cluster dynamically.

Automatic removal of dynamically added hosts

By default, dynamically added hosts remain in the cluster permanently. Optionally, you can use LSF_DYNAMIC_HOST_TIMEOUT in `lsf.conf` to set an optional timeout value in hours or minutes.

LSF_DYNAMIC_HOST_TIMEOUT (lsf.conf)

If LSF_DYNAMIC_HOST_TIMEOUT is defined and a host is not a master candidate, when the host is unavailable for longer than the value specified, it is removed from the cluster.

Adding 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 29 characters long.

Adding 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
CRAYJ
CRAYC
CRAYT
DigitalUNIX
HPPA
IBMAIX4
SGI6
SUNSOL
SONY
WIN95
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 94.

```
Begin HostModel
MODELNAME  CPUFACTOR   ARCHITECTURE # keyword
# x86 (Solaris, NT, Linux): approximate values, based on SpecBench results
# for Intel processors (Sparc/NT) 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 `ypcat 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_LIM_PORT=3879
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

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

Configuring services manually

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.

1. Log on to any host as `root`.
2. 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
```

3. Run `lsadmin reconfig` to reconfigure LIM.
4. Run `badmin reconfig` to reconfigure `mbatchd`.
5. 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.

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 `ypcat 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 `ypbind` 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*. 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.

Sun's 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.

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
```

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.

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

Configuring 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 `badmin 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 `badmin reconfig` if you do not want the new group to be recognized by jobs that were submitted before you reconfigured.
   b. Run `badmin mbdrestart` if you want the new host to be recognized by jobs that were submitted before you reconfigured.

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`.

Viewing normalized ratings

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.

Tuning 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 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.

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.

Host exceptions LSF can detect

If you configure 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`).

By default, 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.

Default eadmin actions

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.

Configuring host exception handling lsb.hosts)

EXIT_RATE Specifies a threshold for exited jobs. If the job exit rate is exceeded for 10 minutes or the period specified by JOB_EXIT_RATE_DURATION, LSF invokes `eadmin` to trigger a host exception.

Example The following Host section defines a job exit rate of 20 jobs per minute for all hosts:

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

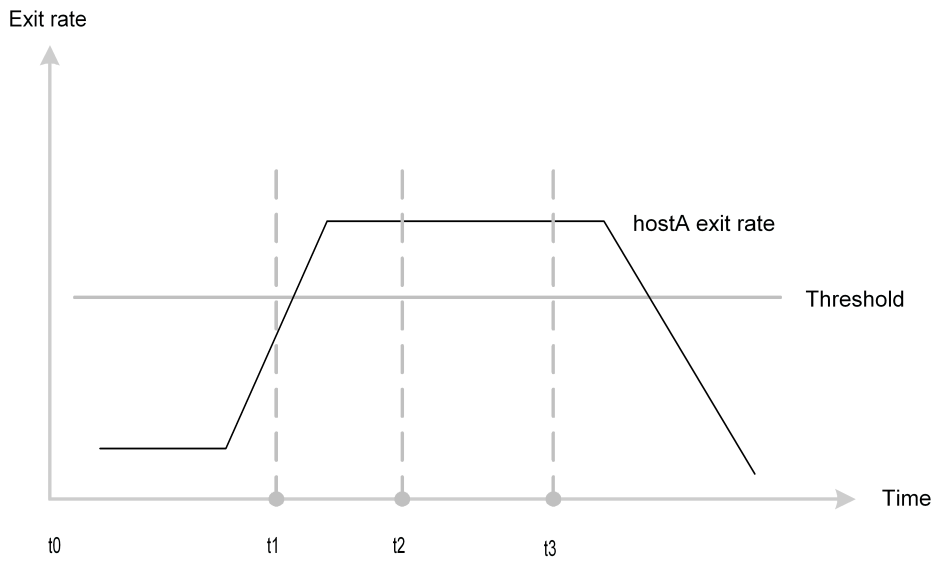
Configuring thresholds for exception handling

JOB_EXIT_RATE_DURATION (lsb.params)

By default, LSF checks the number of exited jobs every 10 minutes. Use JOB_EXIT_RATE_DURATION in `lsb.params` to change this default.

Tuning Tune JOB_EXIT_RATE_DURATION carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

Example



In the diagram, the job exit rate of `hostA` exceeds the configured threshold. LSF monitors `hostA` from time t1 to time t2 (t2=t1 + 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.

C H A P T E R

5

Working with Queues

Contents

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 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	bqueues
Queue status	bqueues
Detailed queue information	bqueues -l
State change history of a queue	badmin qhist
Queue administrators	bqueues -l for queue

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

Viewing available queues and queue status

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.

Viewing detailed queue information

To see the complete status and configuration for each queue, run bqueues -l. You can specify queue names on the command-line to select specific queues. In the example below, more detail is requested 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  -   -   -    -    -

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
```

Viewing the state change history of a queue

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

```
% badmin 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>.
```

Viewing queue administrators

Use `bqueues -l` for the queue.

Viewing exception status for queues (bqueues)

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    -     -     -     -       -     -    -     -     -      -      -

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
```

Controlling Queues

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

Closing a queue

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
```

Opening a queue

Run `badmin qopen`:

```
% badmin qopen normal
Queue <normal> is opened
```

Inactivating a queue

Run `badmin qinact`:

```
% badmin qinact normal
Queue <normal> is inactivated
```

Activating a queue

Run `badmin qact`:

```
% badmin qact normal
Queue <normal> is activated
```

Logging a comment when controlling a queue

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

```
% 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" "6.0 1050082373 1 "normal" 32185 "lsfadmin" "add user"
"QUEUE_CTRL" "6.0 1050082380 1 "normal" 32185 "lsfadmin" "add user user1"
```

Use `badmin hist` or `badmin qhist` to display administrator comments for closing and opening hosts. For example:

```
% badmin 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.
Th
is 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    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    -     -     -     -       -     -    -     -     -      -      -

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"
```

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, dispatch windows are not configured, queues are always Active.

To configure dispatch window:

1. Edit `lsb.queues`
2. Create a DISPATCH_WINDOW keyword for the queue and specify one or more time windows. For example:

   ```
   Begin Queue
   QUEUE_NAME    = queue1
   PRIORITY      = 45
   DISPATCH_WINDOW = 4:30-12:00
   End Queue
   ```

3. Reconfigure the cluster using:
   a. `lsadmin reconfig`
   b. `badmin reconfig`
4. Run `bqueues -l` to display the dispatch windows.

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, run windows are not configured, queues are always Active and jobs can run until completion.

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. For example:

   ```
   Begin Queue
   QUEUE_NAME    = queue1
   PRIORITY      = 45
   RUN_WINDOW = 4:30-12:00
   End Queue
   ```

3. Reconfigure the cluster using:
   a. `lsadmin reconfig.`
   b. `badmin reconfig.`
4. Run `bqueues -l` to display the run windows.

Adding and Removing Queues

Adding 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.

   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.

Removing 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.

Steps

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. For example:

```
% 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
SUBMIT_TIME
5308  user5  RUN   night    hostA     hostD          job5   N
ov 21 18:16
5310  user5 PEND   night    hostA     hostC         job10   N
ov 21 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`.

Managing Queues

Restricting 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 `badmin ckconfig` to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
5. Run `badmin 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 `badmin mbdrestart`. For more details on `badmin mbdrestart`, see "Reconfiguring Your Cluster" on page 65.

Adding 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.

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. For example:

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

Handling Job Exceptions

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.

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, 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`. Alternatively, `eadmin` could find out the owner of the problem jobs and use `bstop -u` to stop all jobs that belong to the user.

Job exceptions LSF can detect

If you configure exception handling, 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 5 minutes. 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 5 minutes. Use EADMIN_TRIGGER_DURATION in `lsb.params` to change how frequently LSF checks for idle jobs.

Default eadmin actions

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 10 minutes, and 1 overrun job and 2 idle jobs are detected, after 10 minutes, `eadmin` is invoked and only one email is sent. If another overrun job is detected in the next 10 minutes, another email is sent.

Configuring job exception handling (lsb.queues)

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

JOB_IDLE Specifies 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 `eadmin` to trigger the action for a job idle exception.

JOB_OVERRUN Specifies a threshold for job overrun. If a job runs longer than the specified run time, LSF invokes `eadmin` to trigger the action for a job overrun exception.

JOB_UNDERRUN Specifies a threshold for job underrun. If a job exits before the specified number of minutes, LSF invokes `eadmin` to trigger the action for a job underrun exception.

Example The following queue defines thresholds for all 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

EADMIN_TRIGGER_DURATION (lsb.params)

By default, LSF checks for job exceptions every 5 minutes. Use EADMIN_TRIGGER_DURATION in `lsb.params` to change how frequently LSF checks for overrun, underrun, and idle jobs.

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

C H A P T E R

6

Managing Jobs

Contents

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

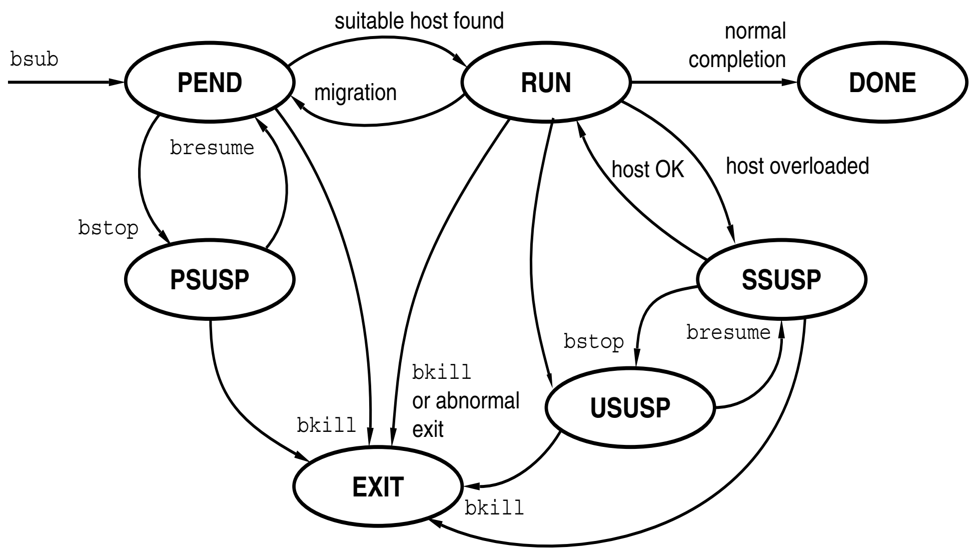
Suspended job states

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.



Viewing running jobs

Use the `bjobs -r` command to display running jobs.

Viewing done jobs

Use the `bjobs -d` command to display recently completed jobs.

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

Viewing pending reasons

Use the `bjobs -p` command to display the reason why a job is pending.

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.

Viewing suspension reasons

Use the `bjobs -s` command to display the reason why a job was suspended.

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.

Viewing wait status and wait reason

Use the `bhist -l` command 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.

See Chapter 24, "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.

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 96 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. The post-processing of a repetitive job cannot be longer than the repetition period.

Viewing post-execution states

Use the `bhist` command to display the POST_DONE and POST_ERR states. The resource usage of post-processing is not included in the job resource usage.

Chapter 28, "Pre-Execution and Post-Execution Commands" for more information.

Viewing 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.

Viewing all jobs for all users

Run `bjobs -u all` to display all jobs for all users. Job information is displayed in the following order:

1. Running jobs
2. Pending jobs in the order in which they will be scheduled
3. 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_TIM
E
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
```

Viewing jobs for specific users

Run `bjobs -u` *`user_name`* to display jobs for a specific user. For example:

```
% bjobs -u user1
JOBID   USER    STAT    QUEUE     FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIM
E
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
```

Viewing exception status for jobs (bjobs)

Use `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    -     -     -     -       -     -    -     -     -      -      -

 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    -     -     -     -       -     -    -     -     -      -      -

 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
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
```

Switching 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.

Switching a single job

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
```

Switching all jobs

Run `bswitch -q` *`from_queue to_queue`* `0` to switch all the jobs in a queue to another queue. The example below selects jobs from the `night` queue and switches them to the `idle` 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:

```
% 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.

Forcing a pending job to run

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.

Suspending a job

Run `bstop` *`job_ID`*. Your job goes into `USUSP` state if the job is already started, or into `PSUSP` state if it is pending. For example:

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

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`.

Windows `bstop` causes the job to be suspended.

Resuming a job

Run `bresume` *`job_ID`*. For example:

```
% 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.

Killing a job

Run `bkill` *`job_ID`*:

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

kills job 3421.

Forcing removal of a job from LSF

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.

Sending a signal to a job

Run `bkill -s` *`signal job_id`*, where *`signal`* is either the signal name or the signal number. For example:

```
% bkill -s TSTP 3421
Job <3421> is being signaled
```

sends the `TSTP` signal to job 3421.

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.

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.

Creating a job group

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`

  creates a job group named `risk_group` under the root group `/`.
- `% bgadd /risk_group/portfolio1`

  creates a job group named `portfolio1` under job group `/risk_group`.
- `% bgadd /risk_group/portfolio1/current`

  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.

Submitting jobs under a job group

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. 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.

-g and -sla options

You cannot use the `-g` option with `-sla`. A job can either be attached to a job group or a service class, but not both.

Viewing jobs in job groups

bjgroup command

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

```
% bjgroup
GROUP_NAME          NJOBS    PEND     RUN     SSUSP  USUSP  FINISH
/fund1_grp            5        4       0        1      0      0
/fund2_grp           11        2       5        0      0      4
/bond_grp             2        2       0        0      0      0
/risk_grp             2        1       1        0      0      0
/admi_grp             4        4       0        0      0      0
```

bjobs command

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>;
...
```

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
```

Modifying (bmod)

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 operate on finished, running, and pending 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);
...
```

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.

C H A P T E R

7

Managing Users and User Groups

Contents

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
- 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.

Viewing user information

Run `busers all`. For example:

```
% 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
```

Viewing user group information

Run bugroup. For example:

```
% bugroup
GROUP_NAME           USERS
testers              user1 user2
engineers            user3 user4 user10 user9
develop              user4 user10 user11 user34 engineers/
system               all users
```

Viewing user share information

Run `bugroup -l`, which displays user share group membership information in long format. For example:

```
% 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.

Configuring user groups

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

```
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 `badmin ckconfig` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
9. Run `badmin 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`.

P A R T

II

Working with Resources

Contents

C H A P T E R

8

Understanding Resources

Contents

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.

Viewing available resources

lsinfo 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
ALPHA
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
```

lshosts Use `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 ()
```

Viewing host load by resource

lshosts Use `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 `lsload -s` command displays dynamic shared resources.

How Resources are Classified

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	Resources defined by user sites: external load indices and resources defined in the lsf.shared 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. For example, 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

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.

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.

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.

Viewing shared resources for hosts

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.

Viewing load on a host

Use `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    253M  297M
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   me
m
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
name	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 lshosts to display run windows.
`lockU`	The host is locked by an LSF administrator or root.

Status	Description
`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 NT, 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 NT

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
`type`	host type	string	configuration
`model`	host model	string	configuration
`hname`	host name	string	configuration
`cpuf`	CPU factor	relative	configuration
`server`	host can run remote jobs	Boolean	configuration
`rexpri`	execution priority	`nice`(2) argument	configuration
`ncpus`	number of processors	processors	LIM
`ndisks`	number of local disks	disks	LIM
`maxmem`	maximum RAM	MB	LIM
`maxswp`	maximum swap space	MB	LIM
`maxtmp`	maximum space in `/tmp`	MB	LIM

CPU factor (cpuf)

The CPU factor is the speed of the host's CPU relative to other hosts in the cluster. If one processor is twice the speed of another, its CPU factor should be twice as large. The CPU factors are defined by the LSF administrator. 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.

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

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+
- Compaq Alpha 5.0+
- 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 * 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*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.

C H A P T E R

9

Adding Resources

Contents

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.

Adding New Resources to Your Cluster

To add host resources to your cluster, use the following steps:

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 154.
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 lsf.cluster.cluster_name ResourceMap Section" on page 155.
5. Reconfigure your cluster.

Configuring lsf.shared Resource Section

Configured resources are defined in the `Resource` section of `lsf.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 names:

  `cpu cpuf io login ls idle maxmem maxswp maxtmp type model status it mem ncpus ndisks pg r15m r15s r1m swap swp tmp ut`
- Resource names are case sensitive
- Resource names can be up to 29 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
```

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).

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 under LSF Base.

- 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.
- The RESOURCES column in the `Host` section of the `lsf.cluster.`*`cluster_name`* file should be used to associate static Boolean resources with particular hosts.
- Almost all 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 `swap`. They are viewed using `lsload -l` or `lsload -I`.

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

To increase LIM reliability, LSF Version 6.0 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 160.
3. Put your ELIM in LSF_SERVERIR.

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

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

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 stared 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 LIM 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.mysrc`) sets the value of `myrsc` 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 "myrsc"
   val=2

   # create an output string in the format:
   # number_indices index1_name index1_value...

   reportStr="1 myrsc $val"
   echo "$reportStr"

   # wait for 30 seconds before reporting again
   sleep 30
done
```

2 Test this ELIM by running it from the command line.

```
% ./elim.myrsc
```

It should give you the output:

```
1 myrsc 2
```

3 Copy the ELIM to LSF_SERVERDIR and make sure it has the name `elim.myrsrc`.

4 Define the `myrsc` 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
myrsc         Numeric  30        N     (custom resource to trigger elim to
start up)
End Resource
```

5 Map the `myrsc` resource to hosts in `lsf.cluster.`*`cluster_name`*. In this case, we want this resource to reside only on `hostA`.

```
Begin ResourceMap
RESOURCENAME          LOCATION
myrsc                [hostA]
End ResourceMap
```

6 Reconfigure LSF with the commands:
   - `lsadmin reconfig`
   - `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  myrsc
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.

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 LIM 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 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.

Steps

For example, some sites prefer to use `/usr/tmp` for temporary files.

To 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 160.

   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
```

P A R T

III

Scheduling Policies

Contents

C H A P T E R

10

Time Syntax and Configuration

Contents

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 (Sunday) to 6 (Saturday), representing the day of the week.

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 = time1-time2
```

Time 1 is the start of the window and time 2 is the end of the window. Both time values must use the same syntax. There are 3 different ways to specify a time window:

- *hour-hour*
- *hour*:*minute*-*hour*:*minute*
- *day*:*hour*:*minute*-*day*:*hour*:*minute*

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. For more details on time windows, see "Specifying Time Windows" on page 169.

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)
```

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.users`

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 `badmin 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.

lsb.hosts example

In the following example, the `#if`, `#else`, `#endif` are not interpreted as comments by LSF but as if-else constructs.

```
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.queues example

```
Begin Queue
...
#if time(8:30-18:30)
   INTERACTIVE  = ONLY  # interactive only during day shift
#endif
...
End Queue
```

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 170.

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
```

Verifying configuration

Use the following LSF commands to verify configuration:

- `bparams(1)`
- `busers(1)`
- `bhosts(1)`
- `bqueues(1)`

C H A P T E R

11

Deadline Constraint and Exclusive Scheduling

Contents

Deadline Constraint Scheduling

Deadline constraints

Deadline constraints will suspend or terminate running jobs at a certain time. There are 2 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.

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
```

Exclusive Scheduling

About 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.

An exclusive job cannot preempt another job, and cannot be preempted by another job.

Configuring an exclusive queue

To configure an exclusive queue, set EXCLUSIVE in the queue definition (`lsb.queues`) to `Y`.

Configuring a host to run one job at a time

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`.

Submitting an exclusive job

To submit an exclusive job, use the `-x` option of `bsub` and submit the job to an exclusive queue.

C H A P T E R

12

Preemptive Scheduling

Contents

About Preemptive Scheduling

Preemptive scheduling lets a pending high-priority job take resources away from a running job of lower priority. When 2 jobs compete for the same resource, LSF automatically suspends the low-priority job to make resources 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.

Limitation The following types of jobs cannot be preempted:

- Jobs that have been forced to run with the command `brun`
- NQS jobs
- Backfill jobs
- Exclusive 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 resource. Their queue must be able to preempt the low-priority queue.

Preemptable jobs Preemptable jobs are running in a low-priority queue and are holding the specified resource. Their queue must be able to be preempted by the high-priority queue.

How Preemptive Scheduling Works

Preemptive scheduling occurs when two jobs compete for the same resource. If a high-priority job is pending, LSF can suspend a lower priority job that is running, and then start the high-priority job using the job slot that becomes available. For this to happen, the high-priority job must be pending in a preemptive queue, or the low-priority job must belong to a preemptable queue.

By default, when multiple preemptable jobs exist (low-priority jobs holding the required resource), LSF preempts a job from the least-loaded host.

The preempted job is resumed as soon as more resources become available; it does not necessarily have to wait for the preempting job to finish.

Queues that can preempt others are more aggressive in scheduling jobs because a resource that is not available to a low-priority queue might be available (by preemption) to a high-priority queue.

By default, job slot limits are enforced based on the number of job slots taken by running and suspended jobs. With preemptive scheduling, the suspended jobs don’t count against certain job slot limits. This means that when one job is suspended, another job can be started in its place.

LSF makes sure that the number of running jobs never exceeds the job slot limits. When LSF tries to place a preemptive job, LSF considers each job slot limit, but for certain job slot limits, LSF only counts the job slots used by running jobs that are not preemptable. Then, if starting the preemptive job would violate job slot limits, LSF suspends one or more low-priority jobs.

Job slot limits affected by preemptive scheduling

When you enable preemptive scheduling, you automatically affect the following job slot limits:

- Total job slot limit for hosts, specified at the host level (SLOTS and HOSTS in `lsb.resources`)
- Total job slot limit for individual users, specified at the user level (SLOTS and USERS in `lsb.resources`); by default, suspended jobs still count against the limit for user groups

You can configure preemptive scheduling to affect following limits:

- Total job slot limit for user groups, specified at the user level (SLOTS and USERS in `lsb.resources`); if preemptive scheduling is enabled, suspended jobs never count against the limit for individual users
- Total number of jobs for users and user groups, specified at the host level (SLOTS, PER_USER=all, and HOSTS in `lsb.resources`)
- Per-processor job slot limit for individual users, specified at the user level (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in `lsb.resources`)
- Per-processor job slot limit for user groups, specified at the user level (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in `lsb.resources`)

Job slot limits specified at the queue level are never affected by preemptive scheduling; they are always enforced for both running and suspended jobs.

Preemption of multiple job slots

If multiple resources are required, LSF can preempt multiple jobs, until sufficient resources are available. For example, one or more jobs might be preempted for a job that needs multiple job slots.

Preemption of 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.

Use the PREEMPT_FOR parameter in `lsb.params` to enable the optimized preemption of parallel jobs, so LSF preempts fewer of the low-priority parallel jobs.

See “Optimized Preemption of Parallel Jobs” on page 460 for more information.

Configuring Preemptive Scheduling

To configure preemptive scheduling, make at least one queue in the cluster preemptive (not the lowest-priority queue) or preemptable (not the highest-priority queue).

To make a queue preemptive or preemptable, set PREEMPTION in the queue definition (`lsb.queues`) to PREEMPTIVE or PREEMPTABLE. A queue can be both: its jobs can always preempt jobs in lower priority queues, and can always be preempted by jobs from higher priority queues.

Syntax

PREEMPTION = PREEMPTIVE[[*queue_name*[+*pref_level*]...]]
PREEMPTABLE[[*queue_name*...]]

When you specify a list of queues, you must enclose the list in one set of square brackets.

PREEMPTIVE[[*queue_name*[+*pref_level*] ...]]

Defines a preemptive queue. Jobs in this queue can preempt jobs from specified lower priority queues only, or from all lower priority queues by default if the parameter is specified with no queue names.

If you specify a list of lower-priority preemptable queues, you must enclose the list in one set of square brackets. To indicate an order of preference for the lower-priority queues, put a plus sign (+) after the names of queues and a preference level as a positive integer. See "Configuring preemptable queue preference" on page 182 for more information.

PREEMPTABLE [[*queue_name* ...]]

Defines a preemptable queue. Jobs in this queue can be preempted by jobs from specified higher-priority queues, or from all higher-priority queues by default (if the parameter is specified with no queue names), even if the higher-priority queues are not preemptive.

If you specify a list of higher-priority queues, you must enclose the list in one set of square brackets.

Example

In these examples, assume Queue 1 has highest priority and Queue 4 the lowest.

- If the following settings are in `lsb.queues`:

  ```
  QUEUE_NAME=Queue1
  PREEMPTION=PREEMPTIVE
  ```

  Queue 1 is preemptive, so it can preempt jobs in all lower-priority queues (Queues 2, 3 and 4).

- If the following settings are in `lsb.queues`:

  ```
  QUEUE_NAME=Queue1
  PREEMPTION=PREEMPTIVE[Queue3 Queue4]
  ```

  Queue 1 is preemptive, but can only preempt jobs in Queues 3 and 4, not Queue 2.

- If the following settings are in `lsb.queues`:

  ```
  QUEUE_NAME=Queue3
  PREEMPTION=PREEMPTIVE PREEMPTABLE
  ```

Queue 3 is preemptive and preemptable, so it can preempt jobs in all lower-priority queues (Queue 4), and its jobs can be preempted by all higher-priority queues (Queues 1 and 2).

Configuring additional job slot limits for preemptive scheduling

The following job slot limits are always affected by preemptive scheduling:

- Total job slot limit for hosts, specified at the host level (SLOTS and HOSTS in `lsb.resources`)
- Total job slot limit for individual users, specified at the user level (SLOTS and USERS in `lsb.resources`); by default, suspended jobs still count against the limit for user groups

PREEMPT_FOR in lsb.params

To configure additional job slot limits to be affected by preemptive scheduling, set PREEMPT_FOR in `lsb.params`, and use one or more of the following keywords to indicate that suspended jobs do not count against that job slot limit:

- GROUP_MAX—total job slot limit for user groups, specified at the user level (MAX_JOBS in `lsb.users`); if preemptive scheduling is enabled, suspended jobs never count against the limit for individual users
- HOST_JLU—total number of jobs for users and user groups, specified at the host level (JL/U in `lsb.hosts`)
- USER_JLP—user-processor job slot limit for individual users, specified at the user level (JL/P in `lsb.users`)
- GROUP_JLP—per-processor job slot limit for user groups, specified at the user level (JL/P in `lsb.users`)

Job slot limits specified at the queue level are never affected by preemptive scheduling.

Configuring preemptable queue preference

For preemptive queues, you can specify which preemptable queues are considered first for preemption by configuring queue preference in the queue with the highest priority.

To indicate the order of preference for the preemptable queues, put a plus sign (+) after the names of the preemptable queues and a preference level as a positive integer. Higher numbers indicate higher preferences for preempting a job in that queue. If no queue preference is specified, it is assumed to be 0. If there are multiple queues, LSF preempts jobs in the queue with the highest preference; queues at the same level of preference are ordered by queue priority.

When preemtable queue preference is enabled, LSF considers jobs from preferred queues first instead of choosing running jobs based on best hosts:

- Queues with a preference number are preferred over queues without a preference number
- Queues with a higher preference number are preferred over queues with lower preference number

- For queues without a preference number, the queue with lower-priority is preferred than the queue with higher priority

Example

```
Begin Queue
QUEUE_NAME       = high_priority
PRIORITY         = 50
PREEMPTION       = PREEMPTIVE[low_q1+2   low_q2    low_q3]
End Queue
```

Jobs from `low_q1` are preferred first for preemption before jobs from `low_q2` and `low_q3`.

If preemptable queue preference and preemption on jobs with least run time are both enabled, the queue preference for the job is considered first, then the job run time.

Preempting jobs with the least run time

By default, when more than one preemptable job exists (low-priority jobs holding the required resource), LSF preempts a job from the least-loaded host.

You can configure LSF to consider the jobs with least run time first instead of choosing jobs based on least-loaded host. Run time is wall-clock time, not normalized run time.

If preemptable queue preference and preemption on jobs with least run time are both enabled, the queue preference for the job is considered first, then the job run time.

Configuring preemption on least run time

Set PREEMPT_FOR in `lsb.params`, and use the LEAST_RUN_TIME keyword to indicate that jobs with the least run time are to be preempted before other jobs.

Preventing preemption by run time

You can configure LSF to prevent preemption of a job that would finish within a specified time or that has been running for a specified time. Run time is wall-clock time, not normalized run time.

You must define a run limit for the job, either at job level by `bsub -W` option or in the queue by configuring RUNLIMIT in `lsb.queues`.

NO_PREEMPT_RUN_TIME, NO_PREEMPT_FINISH_TIME (lsb.params)

Set NO_PREEMPT_RUN_TIME in `lsb.params`, and the jobs have been running for the specified number of minutes or longer will not be preempted.

Set NO_PREEMPT_FINISH_TIME in `lsb.params`, and jobs that will finish within the specified number of minutes will not be preempted.

CHAPTER

13

Specifying Resource Requirements

Contents

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==any`). 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 individual applications, or for individual jobs.

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.

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 resource requirement is specified, the queue-level resource requirements become the default resource requirements for the job.

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==ALPHA && r1m < 2.0)||(type==HPPA && r1m < 1.0))]
```

This will allow a queue, which contains ALPHA and HPPA 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. There are two types of load thresholds:

loadSched The scheduling threshold which 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 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 27, "Load Thresholds" for information about suspending conditions and configuring load thresholds.

Viewing queue-level resource requirements

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.

Example
```
% bsub -R "swp > 15 && hpux order[cpu]" 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.

Viewing job-level resource requirements

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.
...
```

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;
...
```

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
- `lsload` selects and orders hosts
- `lsplace` uses the information in `select`, `order`, and `rusage` sections to select an appropriate host for a task
- `lsloadadj` 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] ...] span[span_string]
same[same_string]
```

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.

How queue-level and job-level requirements are resolved

If job-level resource requirements are specified together with queue-level resource requirements:

- 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`.
- 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.

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`, `lsplace`, `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`.

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
-a	Negative of `a`
!a	Logical not: `1` if `a==0`, `0` otherwise
a * b	Multiply `a` and `b`
a / b	Divide `a` by `b`
a + b	Add `a` and `b`
a - b	Subtract `b` from `a`
a > b	`1` if `a` is greater than `b`, `0` otherwise
a < b	`1` if `a` is less than `b`, `0` otherwise
a >= b	`1` if `a` is greater than or equal to `b`, `0` otherwise
a <= b	`1` if `a` is less than or equal to `b`, `0` otherwise
a == b	`1` if `a` is equal to `b`, `0` otherwise
a != b	`1` if `a` is not equal to `b`, `0` otherwise
a && b	Logical AND: `1` if both `a` and `b` are non-zero, `0` otherwise
a || b	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 which 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 will only be scheduled if the value of the `"avail_scratch"` resource is more than 200 MB and will go 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. In order to exclude hosts which 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 which cannot access the scratch resource. The LSF administrator configures which hosts do and do not have access to a particular shared resource.

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.

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.

By default, no resources are reserved.

Batch jobs

The resource usage (`rusage`) section can be specified at the job level or with the queue configuration parameter RES_REQ.

Syntax

`rusage``[`*`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`

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.

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.

Specifying multiple usage strings

Use several comma-separated usage strings to define different duration and decay for any number of resources.

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 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 for 2 minutes:

```
% bsub -R "rusage[mem=20:swap=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 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
```

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 Two kinds of `span` string are supported:

- **`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:

  - Default `ptile` value, specified by *n* processors. For example:

    `span[ptile=4]`

    The job requests 4 processors on each available host, regardless of how many processors the host has.

  - Predefined `ptile` value, specified by '!'. For example:

    `span[ptile='!']`

    uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value.

    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. 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.

    - 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.

See "Controlling Processor Allocation Across Hosts" on page 441 for more information about specifying `span` strings.

Same String

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.

When you specify for example, `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`

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.

C H A P T E R

14

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, "Ways to Configure Fairshare" on page 236.

Contents

About Fairshare Scheduling

Fairshare scheduling divides the processing power of the LSF cluster among users and groups to provide fair access to resources.

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.

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 doesn't 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 (5 shares out of 10).
  - 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 (5 shares out of 56).

Dynamic User Priority

About 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 called 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 based on the following information about each user:

- Number of shares assigned to the user
- 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 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.

Default dynamic priority formula

By default, LSF calculates dynamic priority according to the following formula:

dynamic priority = *number_shares* / (*cpu_time* * CPU_TIME_FACTOR + *run_time* * RUN_TIME_FACTOR + (1 + *job_slots*) * 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

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 Fairshare" on page 211.

Host Partition Fairshare

About host partition fairshare

Fairshare policy configured at the host level handles 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.

Viewing host partition information

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)

Example

```
% 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
```

Configuring host partition fairshare scheduling

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.
- Optionally, use the reserved host name `all` to configure a single partition that applies to all hosts in a cluster.
- Optionally, 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

About queue-level fairshare

Fairshare policy configured at the queue level handles 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.

Viewing queue-level fairshare information

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.

Configuring queue-level fairshare

To configure a fairshare queue, define FAIRSHARE in `lsb.queues` and specify a share assignment for all users of the queue.

Syntax **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 Fairshare

Applying the same fairshare policy to several queues

Fairshare policy configured at the queue level handles resource contention across multiple queues.

You can define a fairshare policy that applies 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.

Viewing cross-queue fairshare information

Run `bqueues -l` to know if a queue is part of cross-queue fairshare. The parameter FAIRSHARE_QUEUES 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 will 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.

Examples

```
% 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

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

```
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.

Steps

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 Fairshare

About hierarchical fairshare

For both queue and host partitions, hierarchical 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.

Viewing hierarchical share information for a group

Use `bugroup -l` to find out if you belong to a group, and what the share distribution is.

This command displays all the share trees that are configured, even if they are not used in any fairshare policy.

Example

```
% 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]
```

Viewing hierarchical share information for a host partition

By default, `bhpart` only displays the top level share accounts associated with the partition.

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)

Example

```
% bhpart -r Partition1
HOST_PARTITION_NAME:  Partition1
HOSTS:  HostA

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.

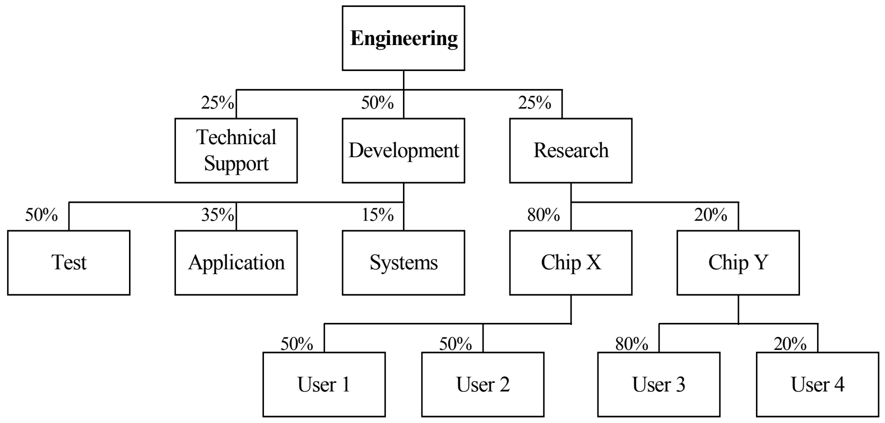
Configuring a share tree

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 hierachical 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.

Example 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 will get 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 x .80 x .25 = .10) and `User3` is entitled to 4% (.80 x .20 x .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 is 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`. 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 fairshare (FAIRSHARE_QUEUES) can undo the dispatching decisions of queue-based fairshare. Cross-queue 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 -> 4 slots
  - `queue3` shares 20% of slots to be allocated = 2 * 6 * 0.2 = 2.4 -> 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 -> 4
  - `queue3` shares 20% of slots to be allocated = 2 * 6 * 0.2 = 2.4 -> 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).

Steps

1. For each queue that uses queue-based fairshare, define the following in `lsb.queues`:
   - SLOT_SHARE
   - SLOT_POOL
2. Optionally, define the following in `lsb.queues` for each queue that uses queue-based fairshare:
   - 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.

     Hosts for queue-based fairshare cannot be in a host partition.
   - 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).

Examples

- The following configures a pool named `poolA`, 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
```

Viewing Queue-based Fairshare Allocations

Viewing configured job slot share

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    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    -     -     -     -       -     -    -     -     -      -      -

USERS:  all users
HOSTS:  groupA/
SLOT_SHARE: 50%
SLOT_POOL: poolA
```

Viewing slot allocation of running jobs

Use `bhosts`, `bmgroup`, and `bqueues` to verify how LSF maintains the configured percentage of running jobs in each queue.

bmgroup command

The queues configurations above use the following hosts groups:

```
% bmgroup -r
GROUP_NAME   HOSTS
groupA       hosta hostb hostc
groupB       hostd hoste hostf
```

bhosts command

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
```

bqueues command

Pool named poolA contains:

- queue1
- queue2
- queue3

poolB contains:

- queue4
- queue5
- queue6

bqueues 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
```

How to interpret the shares

- 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.

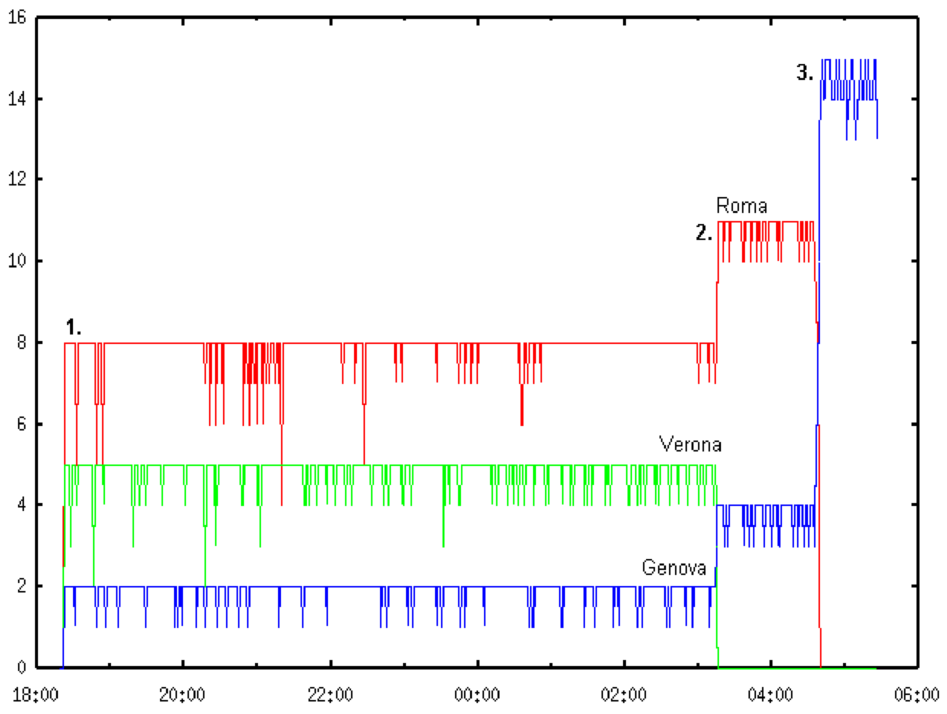
   This leaves 4 slots to be redistributed to queues according to their shares: 40% (2 slots) to `Roma`, 20% (1 slot) to `Genova`. The one remaining slot is assigned to queue `Roma` again:

```
% bqueues
QUEUE_NAME   PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
Roma          50   Open:Active      -    -    -    -   231   221    10     0
Verona        48   Open:Active      -    -    -    -     0     0     0     0
Genova        48   Open:Active      -    -    -    -   496   491     5     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:

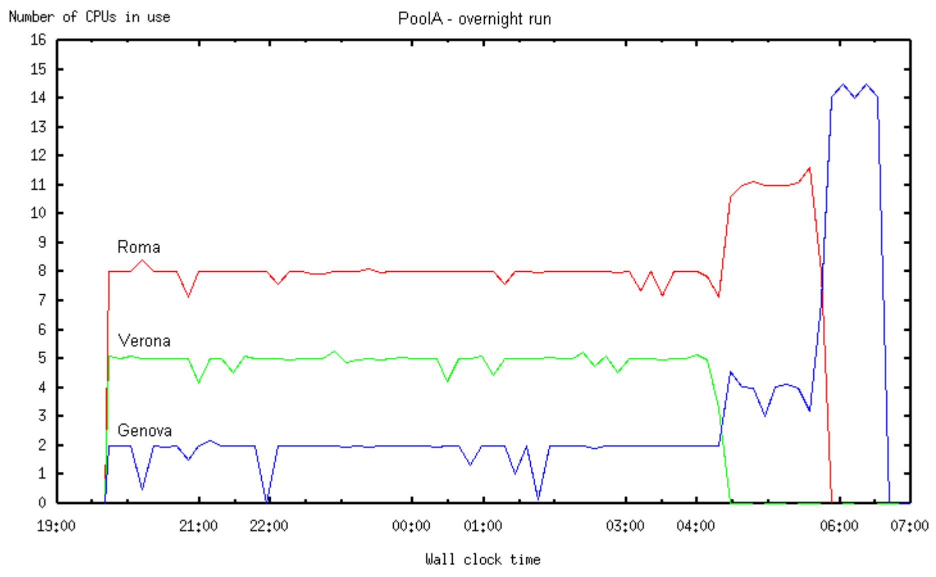


2 pools, 30 job slots, and 2 queues out of any pool

- `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.

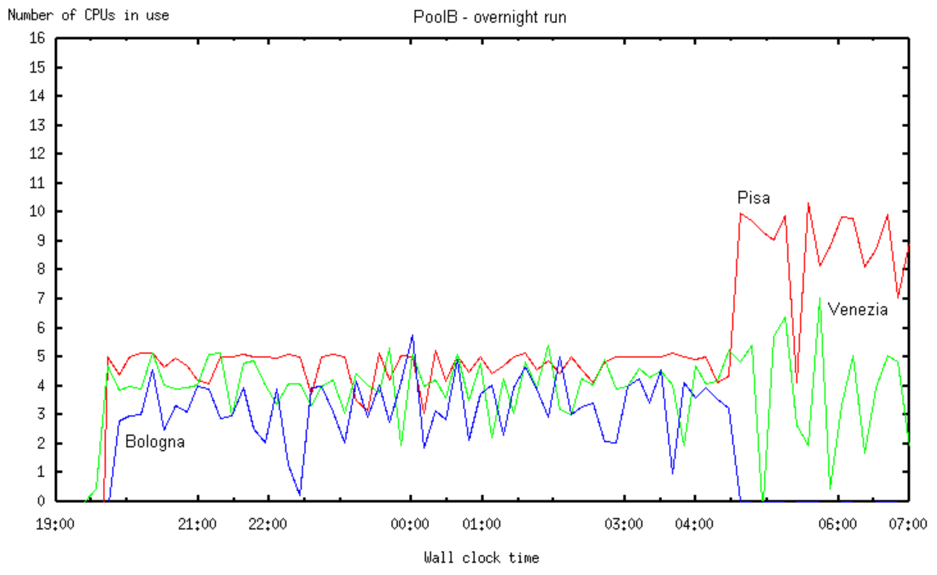
The queues `Milano` and `Parma` run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

```
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:

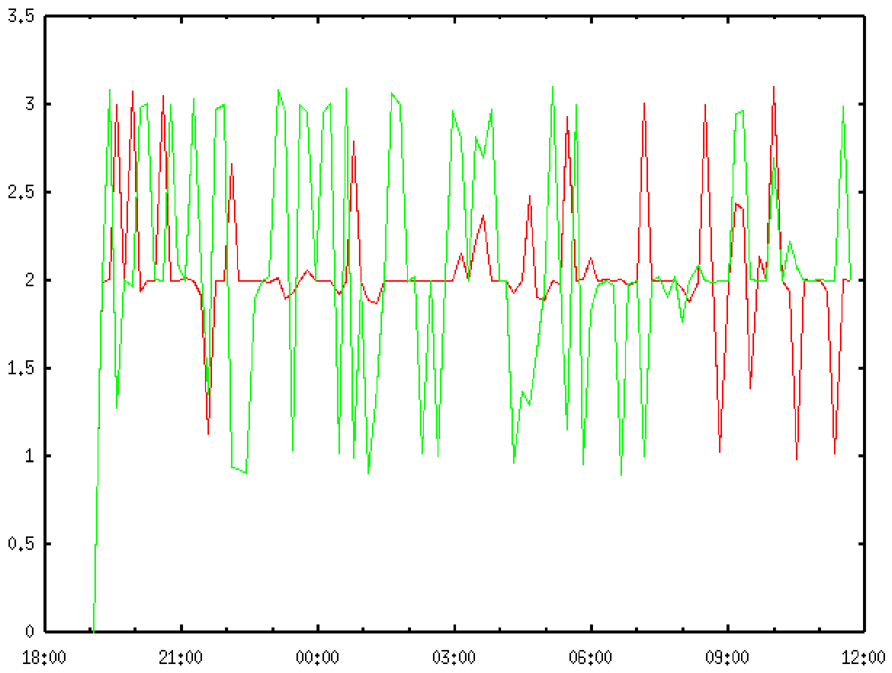
```
% bqueues
QUEUE_NAME    PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
Roma           50   Open:Active      -    -    -    -    15     6    11     0
Verona         48   Open:Active      -    -    -    -    25    18     7     0
Genova         47   Open:Active      -    -    -    -   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
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

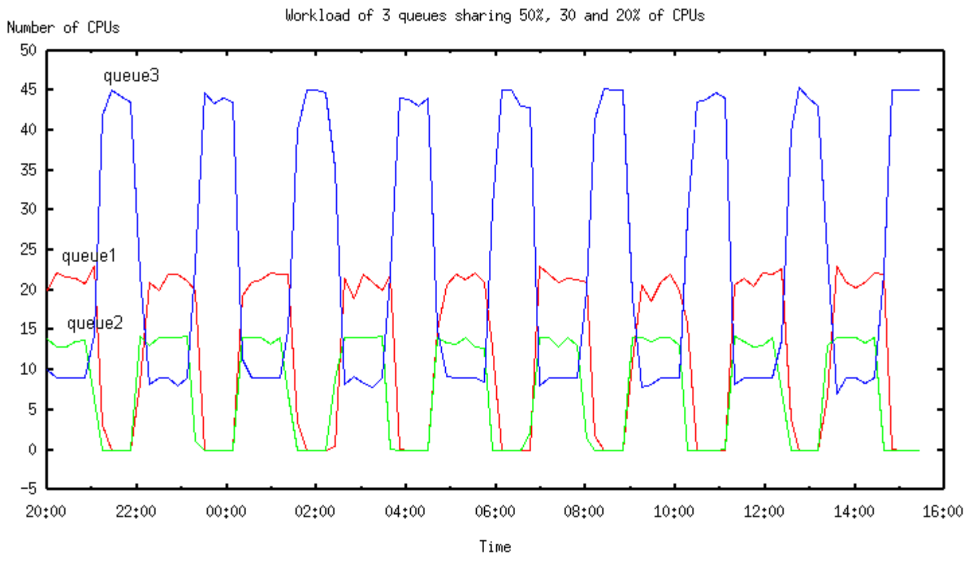
In the following examples, job runtime is not equal, but varies randomly over time.

3 queues in one pool with 50%, 30%, 20% shares

A pool configures 3 queues:

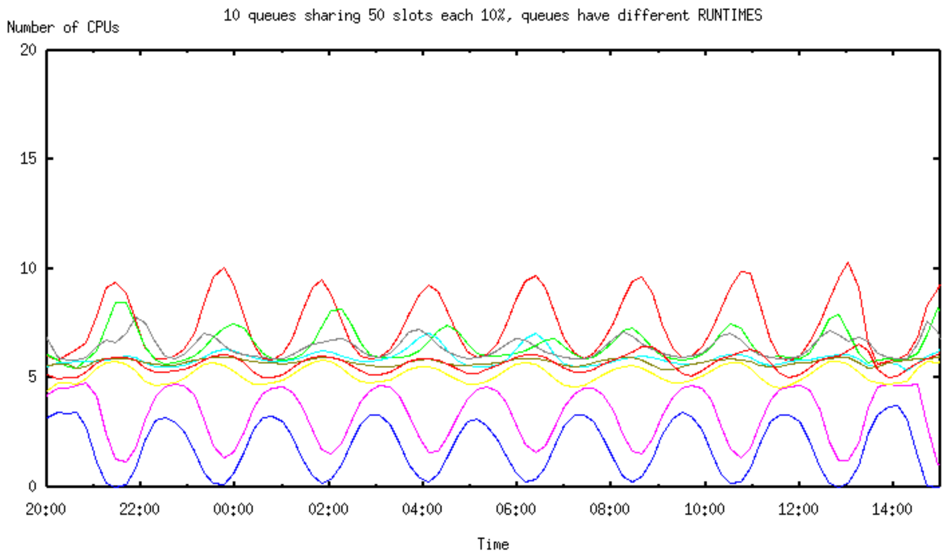
- `queue1` 50% with short-running jobs
- `queue2` 20% with short-running jobs
- `queue3` 30% with longer running jobs

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.

Configuring Specify ENABLE_HST_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:

dynamic priority = *number_shares* / (*cpu_time* * CPU_TIME_FACTOR + (*historical_run_time* + *run_time*) * RUN_TIME_FACTOR + (1 + *job_slots*) * RUN_JOB_FACTOR)

- *historical_run_time*

  The 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:

dynamic priority = *number_shares* / (*run_time* * 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:

dynamic priority = *number_shares* / (*historical_run_time* + *run_time*) * 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.

Configuring

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:

dynamic priority = *number_shares* / (*cpu_time* * CPU_TIME_FACTOR + (*historical_run_time* + *run_time*) * RUN_TIME_FACTOR + (*committed_run_time* - *run_time*) * COMMITTED_RUN_TIME_FACTOR + (1 + *job_slots*) * 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.

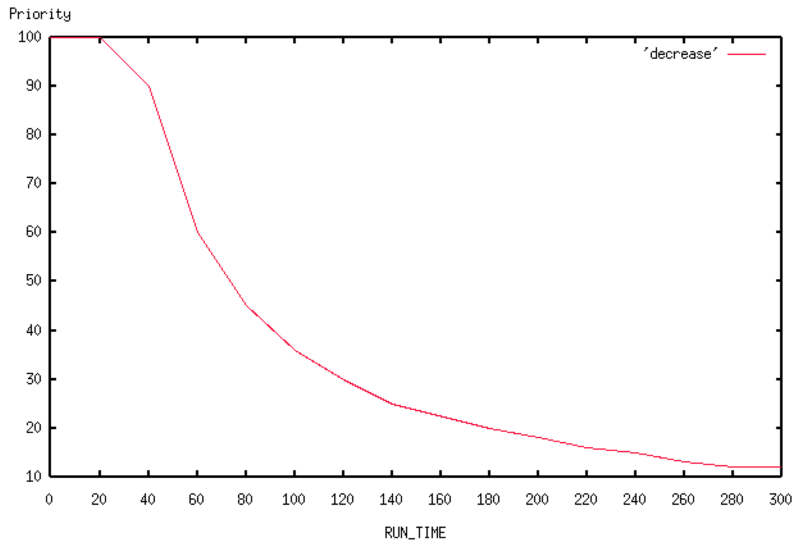
Run time displayed by bqueues and bhpart

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.

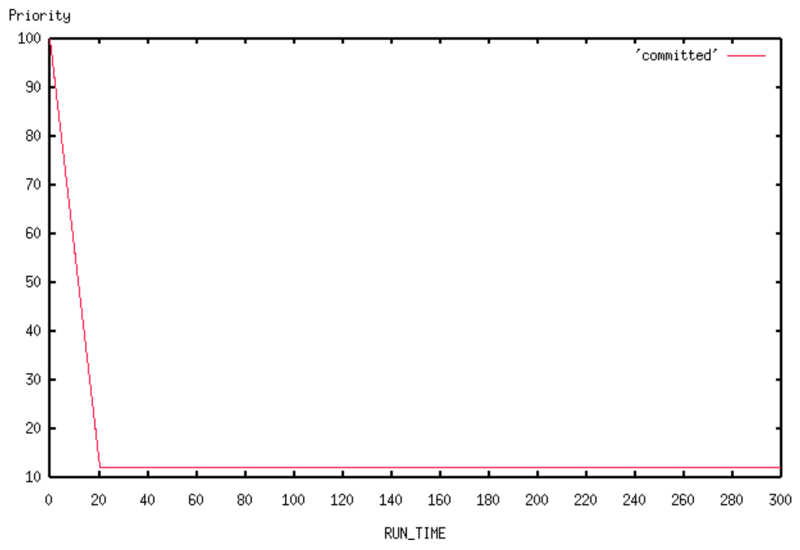
Example The following fairshare parameters are configured in `lsb.params`:

```
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.

Submitting a job and specifying a user group

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.

- `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 2 users compete for resources, their dynamic share priority is the same in every queue.

Configuring

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.

Example

```
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 2 users compete for resources, the most important user is entitled to more resources.

Configuring

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. This is also called round-robin scheduling, because if users submit identical jobs, LSF runs one job from each user in turn.

If 2 users compete for resources, they have equal importance.

Configuring

To configure equal share, use the keyword `default` to define an equal share for every user.

Example

```
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.

- 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.
- 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.

Priority user fairshare

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 2 users compete for resources, and one of them is a priority user, the priority user’s job always runs first.

Configuring

To configure priority users, assign the overwhelming majority of shares to the most important users.

Example A queue is shared by key users and other users. As long as there are jobs from key users waiting for resources, other users' jobs will not be dispatched.

1. Define a user group called `key_users` in `lsb.users`.
2. Configure fairshare and assign the overwhelming majority of shares to the critical users:

```
Begin Queue
QUEUE_NAME = production
FAIRSHARE = USER_SHARES[[key_users@, 2000] [others, 1]]
...
End Queue
```

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.

Static priority fairshare

Static priority fairshare assigns resources to the user with the most shares. Resource usage is ignored.

If 2 users compete for resources, the most important user's job always runs first.

Configuring 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

C H A P T E R

15

Goal-Oriented SLA-Driven Scheduling

Contents

Using Goal-Oriented SLA Scheduling

Goal-oriented 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.

Submitting jobs to a service class

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`.

Submit with run limit

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.

-sla and -g options

You cannot use the `-g` option with `-sla`. A job can either be attached to a job group or a service class, but not both.

Modifying SLA jobs (bmod)

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

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 (the name you use cannot be the same as an existing host partition 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.

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
```

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)] \
        [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
```

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
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

   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 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.

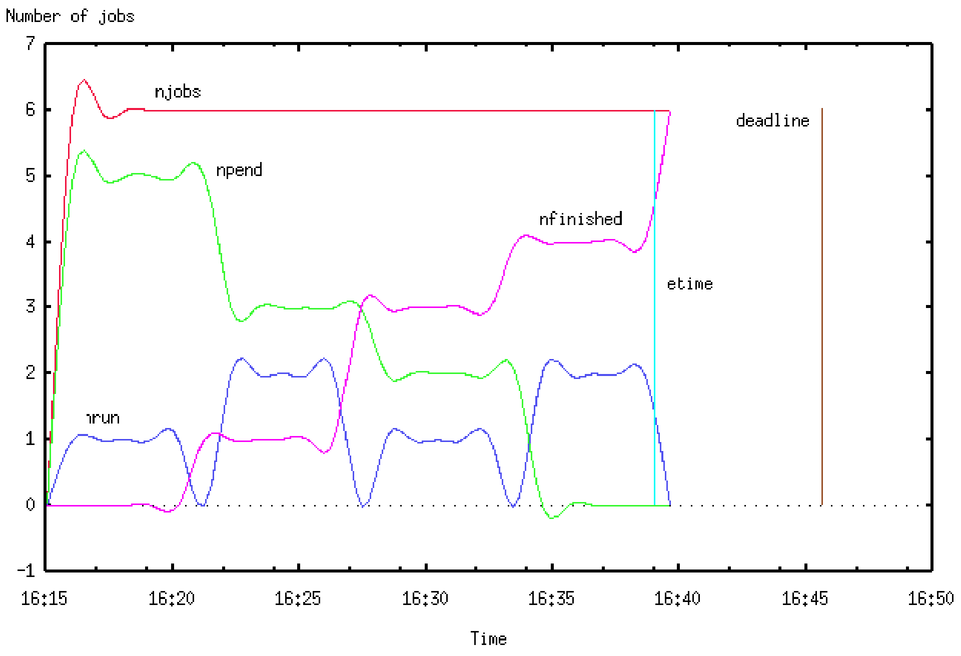
```
Begin ServiceClass
NAME = Quadra
PRIORITY = 20
GOALS = [DEADLINE  timeWindow (16:15-16:45)]
DESCRIPTION = short window
End ServiceClass
```

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.



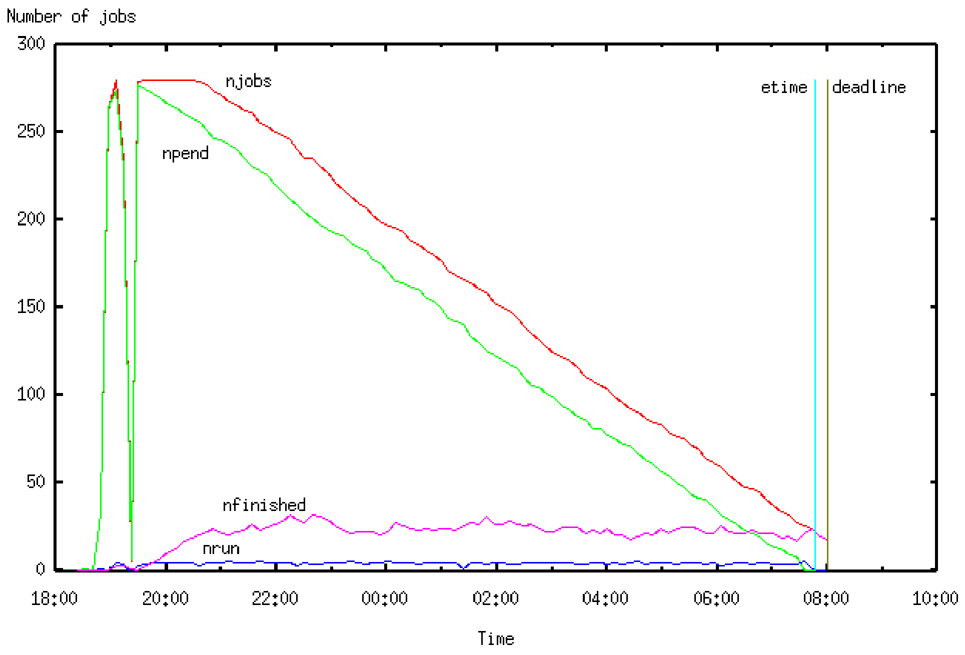
An overnight run with two service classes

`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
```

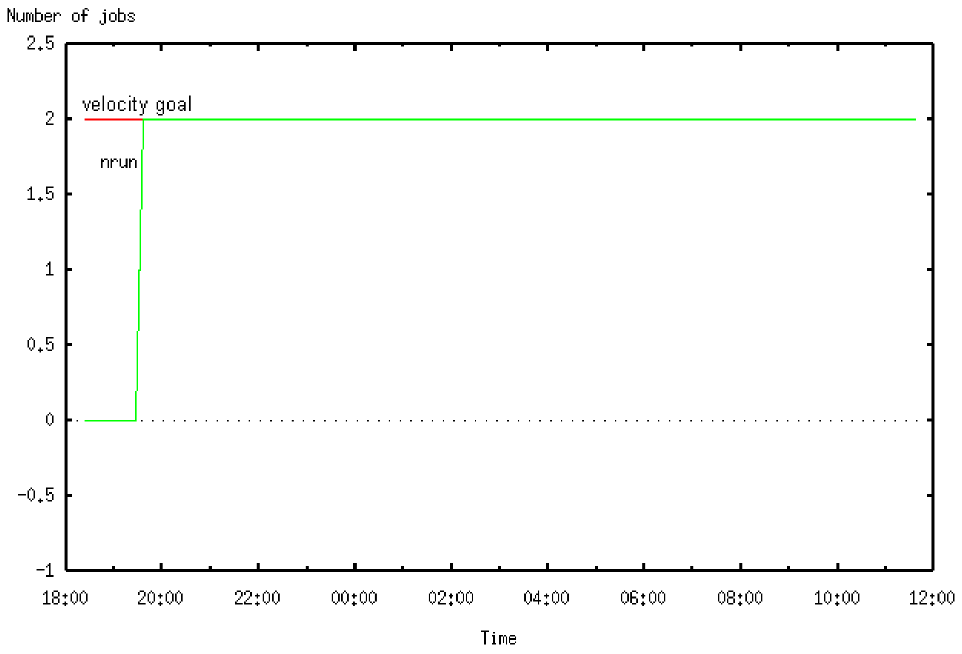
The following illustrates the progress of the deadline SLA `Qualicum` running 280 jobs overnight with random runtimes until the morning deadline. As with the simple deadline goal example, when the finished job curve (`nfinished`) meets the total number of jobs curve (`njobs`) the deadline is met with all jobs completed ahead of the configured deadline.



- `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

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
```

P A R T

IV

Job Scheduling and Dispatch

Contents

C H A P T E R

16

Resource Allocation Limits

Contents

About Resource Allocation Limits

Contents

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 is 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 26, "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
  - 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 often 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.

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.

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

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 Limit section of lsb.resources...
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 lsb.resources...
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

Enabling resource allocation limits

Resource allocation limits scheduling plugin

To enable resource allocation limits in your cluster, configure the resource allocation limits scheduling plugin `schmod_limit` in `lsb.modules`.

Configuring lsb.modules

```
Begin PluginModule
SCH_PLUGIN                RB_PLUGIN
SCH_DISABLE_PHASES
schmod_default                ()                              ()
schmod_limit                  ()                              ()
End PluginModule
```

Configuring 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 6.0 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 6.0 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	Resource consumers (lsb.resources)					Equivalent existing limit (file)
(lsb.resources)	USERS	PER_USER	QUEUES	HOSTS	PER_HOST	
SLOTS	—	**`all`**	—	*host_name*	—	JL/U (lsb.hosts)
SLOTS_PER_PROCESSOR	*user_name*	—	—	—	**`all`**	JL/P (lsb.users)
SLOTS	—	**`all`**	*queue_name*	—	—	UJOB_LIMIT (lsb.queues)
SLOTS	—	**`all`**	—	—	—	MAX_JOBS (lsb.users)
SLOTS	—	—	*queue_name*	—	**`all`**	HJOB_LIMIT (lsb.queues)
SLOTS	—	—	—	*host_name*	—	MXJ (lsb.hosts)
SLOTS_PER_PROCESSOR	—	—	*queue_name*	—	**`all`**	PJOB_LIMIT (lsb.queues)
SLOTS	—	—	*queue_name*	—	—	QJOB_LIMIT (lsb.queues)

Limits for the following resources have no corresponding limit in `lsb.users`, `lsb.hosts`, and `lsb.queues`:

- SWP
- TMP
- LICENSE
- RESOURCE

How conflicting limits are resolved

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
```

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
HOSTS     SLOTS  MEM    SWP  TMP   USERS       QUEUES
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
```

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 host to 1 job slot per processor:

```
Begin Limit
NAME                = default_limit
SLOTS_PER_PROCESSOR = 1
PER_HOST            = all
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)

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 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. Since 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           -        -   10/25         -   10/258
limit1    user1          q3     hostA           -        -       -   30/2953        -
limit1    user1          q4     hostC           -        -       -    40/590        -

EXTERNAL RESOURCE LIMITS:

NAME         USERS    QUEUES   HOSTS   PROJECTS     user1_num     hc_num      HC_num
limit_ext1       -         -   hostA          -             -       1/20           -
limit_ext1       -         -   hostC          -          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.

C H A P T E R

17

Reserving Resources

Contents

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
```

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 = swap>50 rusage[swp=40:duration=5h:decay=1]
.
End Queue
```

Job-level resource reservation

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

Memory Reservation for Pending Jobs

About memory reservation for pending jobs

By default, the `rusage` 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 `rusage` 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.

Configuring memory reservation for pending jobs

RESOURCE_RESERVE parameter

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

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

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
```

Using memory reservation for pending jobs

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 274 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 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.)

Viewing Resource Reservation Information

Viewing host-level resource information

bhosts command 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_WIN
DOW
ok        20.00        -      4        2       1        0        0      1        -

 CURRENT LOAD USED FOR SCHEDULING:
              r15s    r1m   r15m      ut      pg        io     ls      it     tmp        sw
p     mem
 Total         1.5    1.2    2.0    91%      2.5         7     49       0    911M       915
M      0M
 Reserved      0.0    0.0    0.0     0%      0.0         0      0       0      0M         0
M    143M
```

Use `bhosts -s` to view information about shared resources.

Viewing queue-level resource information

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

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

SCHEDULING POLICIES:  RESOURCE_RESERVE

USERS:   all users
HOSTS:   all

Maximum resource reservation time: 600 seconds
```

Viewing reserved memory for pending jobs

bjobs -l

If the job memory requirements cannot be satisfied, `bjobs -l` shows the pending reason. `bjobs -l` shows both reserved slots and reserved memory.

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

C H A P T E R

18

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

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.

In this section

LSF can manage three types of software licenses, described in the following sections:

- "Host Locked Licenses" on page 279
- "Counted Host Locked Licenses" on page 280
- "Network Floating Licenses" on page 281

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 140 for information about configuring Boolean resources.

See the *Platform LSF 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 158 for instructions on writing and using an ELIM and configuring resource requirements for an application.

See the *Platform LSF 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 368 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 301 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 requeuing

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 301 for more information about configuring job requeue
- See Chapter 29, “Job Starters” for more information about LSF job starters

C H A P T E R

19

Dispatch and Run Windows

Contents

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.

Configuring run windows

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 169.

Viewing information about run windows

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.

Configuring dispatch windows

Dispatch windows can be defined for both queues and hosts. The default is no restriction, or always open.

Configuring host dispatch windows

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.

Configuring queue dispatch windows

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.

Displaying dispatch windows

Displaying queue dispatch windows

Use `bqueues -l` to display queue dispatch windows.

Displaying host dispatch windows

Use `bhosts -l` to display host dispatch windows.

CHAPTER

20

Job Dependencies

Contents

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.

Specifying a job dependency

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 292.

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.

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

- 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:

  ```
  bsub -J "JobA" -w 'done(JobB)' command
  ```
- `-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.

C H A P T E R

21

Job Priorities

Contents

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

- "Configuring job priority" on page 296
- "Specifying job priority" on page 297
- "Viewing job priority information" on page 297

Configuring job priority

To configure user-assigned job priority edit `lsb.params` and define MAX_USER_PRIORITY. This configuration applies to all queues in your cluster.

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.

Specifying 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`). Invalid 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`).

Viewing job priority information

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 296) 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.

Configuring job priority escalation

To configure job priority escalation edit `lsb.params` and define JOB_PRIORITY_OVER_TIME. User-assigned job priority must also be configured.

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.

C H A P T E R

22

Job Requeue and Job Rerun

Contents

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

About 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.
- 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.

Configuring automatic job requeue

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.

Example

```
Begin Queue
...
REQUEUE_EXIT_VALUES = 99 100
...
End Queue
```

This configuration enables jobs that exit with 99 or 100 to be requeued.

Reverse Requeue

About 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.

Configuring 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 with the commands `lsadmin reconfig` and `badmin mbdrestart`.

Example LSB_REQUEUE_TO_BOTTOM=1

Exclusive Job Requeue

About 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

Configuring exclusive job requeue

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.

Example

```
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

About 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.

Requeuing a job

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

Examples

```
% 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.

Configuring queue-level job rerun

To enable automatic job rerun at the queue level, set RERUNNABLE in `lsb.queues` to `yes`.

Example

```
RERUNNABLE = yes
```

Submitting a rerunnable job

To enable automatic job rerun at the job level, use `bsub -r`.

Interactive batch jobs (`bsub -I`) cannot be rerunnable.

C H A P T E R

23

Job Checkpoint, Restart, and Migration

Contents

Checkpointing Jobs

Checkpointing a job involves capturing the state of an executing job, the data necessary to restart the job, and not wasting the work done to get to the current stage. The job state information is saved in a checkpoint file. There are many reasons why you would want to checkpoint a job.

Fault tolerance

To provide job fault tolerance, checkpoints are taken at regular intervals (periodically) during the job's execution. If the job is killed or migrated, or if the job fails for a reason other than host failure, the job can be restarted from its last checkpoint and not waste the efforts to get it to its current stage.

Migration

Checkpointing enables a migrating job to make progress rather than restarting the job from the beginning. Jobs can be migrated when a host fails or when a host becomes unavailable due to load.

Load balancing

Checkpointing a job and restarting it (migrating) on another host provides load balancing by moving load (jobs) from a heavily loaded host to a lightly loaded host.

In this section

Approaches to Checkpointing

LSF provides support for most checkpoint and restart implementations through uniform interfaces, `echkpnt` and `erestart`. All interaction between LSF and the checkpoint implementations are handled by these commands. See the `echkpnt(8)` and `erestart(8)` man pages for more information.

Checkpoint and restart implementations are categorized based on the facility that performs the checkpoint and the amount of knowledge an executable has of the checkpoint. Commonly, checkpoint and restart implementations are grouped as kernel-level, user-level, and application-level.

Kernel-level checkpointing

Kernel-level checkpointing is provided by the operating system and can be applied to arbitrary jobs running on the system. This approach is transparent to the application, there are no source code changes and no need to re-link your application with checkpoint libraries.

To support kernel-level checkpoint and restart, LSF provides an `echkpnt` and `erestart` executable that invokes OS specific system calls.

Kernel-level checkpointing is currently supported on:

- Cray UNICOS
- IRIX 6.4 and later
- NEC SX-4 and SX-5

See the `chkpnt(1)` man page on Cray systems and the `cpr(1)` man page on IRIX systems for the limitations of their checkpoint implementations.

User-level checkpointing

LSF provides a method to checkpoint jobs on systems that do not support kernel-level checkpointing called user-level checkpointing. To implement user-level checkpointing, you must have access to your applications object files (.o files), and they must be re-linked with a set of libraries provided by LSF in LSF_LIBDIR. This approach is transparent to your application, its code does not have to be changed and the application does not know that a checkpoint and restart has occurred.

Application-level checkpointing

The application-level approach applies to those applications which are specially written to accommodate the checkpoint and restart. The application writer must also provide an `echkpnt` and `erestart` to interface with LSF. For more details see the `echkpnt(8)` and `erestart(8)` man pages. The application checkpoints itself either periodically or in response to signals sent by other processes. When restarted, the application itself must look for the checkpoint files and restore its state.

Creating Custom echkpnt and erestart for Application-level Checkpointing

Different applications may have different checkpointing implementations and custom `echkpnt` and `erestart` programs.

You can write your own `echkpnt` and `erestart` programs to checkpoint your specific applications and tell LSF which program to use for which application.

- "Writing custom echkpnt and erestart programs" on page 310
- "Configuring LSF to recognize the custom echkpnt and erestart" on page 312

Writing custom echkpnt and erestart programs

Programming language

You can write your own `echkpnt` and `erestart` interfaces in C or Fortran.

Name

Assign the name `echkpnt.`*`method_name`* and `erestart.`*`method_name`*, where *`method_name`* is the name that identifies this is the program for a specific application.

For example, if your custom `echkpnt` is for `my_app`, you would have: `echkpnt.my_app`, `erestart.my_app`.

Location

Place `echkpnt.`*`method_name`* and `erestart.`*`method_name`* in LSF_SERVERDIR. You can specify a different directory with LSB_ECHKPNT_METHOD_DIR as an environment variable or in `lsf.conf`.

The method name (LSB_ECHKPNT_METHOD in `lsf.conf` or as an environment variable) and location (LSB_ECHKPNT_METHOD_DIR) combination must be unique in the cluster. For example, you may have two `echkpnt` applications with the same name such as `echkpnt.mymethod` but what differentiates them is the different directories defined with LSB_ECHKPNT_METHOD_DIR.

The checkpoint method directory should be accessible by all users who need to run the custom `echkpnt` and `erestart` programs.

Supported syntax for echkpnt

Your `echkpnt.`*`method_name`* must recognize commands in the following syntax as these are the options used by `echkpnt` to communicate with your `echkpnt.`*`method_name`*:

```
echkpnt [-c] [-f] [-k | -s] [-d checkpoint_dir] [-x] process_group_ID
```

For more details on `echkpnt` syntax, see the `echkpnt(8)` man page.

Supported syntax for erestart

Your `erestart.`*`method_name`* must recognize commands in the following syntax as these are the options used by `erestart` to communicate with your `erestart.`*`method_name`* .

```
erestart [-c] [-f] checkpoint_dir
```

For more details, see the `erestart(8)` man page.

Return values for echkpnt.*method_name*

If `echkpnt.`*`method_name`* is able to successfully checkpoint the job, it exits with a 0. Non-zero values indicate job checkpoint failed.

`stderr` and `stdout` are ignored by LSF. You can save these to a file by setting LSB_ECHKPNT_KEEP_OUTPUT=y in `lsf.conf` or as an environment variable.

Return values for erestart.*method_name*

`erestart.`*`method_name`* creates the file *`checkpoint_dir`*`/$LSB_JOBID/.restart_cmd` and writes in this file the command to restart the job or process group in the form:

`LSB_RESTART_CMD=`*`restart_command`*

For example, if the command to restart a job is `my_restart my_job`, the `erestart.`*`method_name`* writes to the `.restart_cmd` file:

```
LSB_RESTART_CMD=my_restart my_job
```

`erestart` then reads the `.restart_cmd` file and uses the command specified with LSB_RESTART_CMD as the command to restart the job.

You have the choice of writing to the file or not. Return a 0 if `erestart.`*`method_name`* succeeds in writing the job restart command to the file *`checkpoint_dir`*`/$LSB_JOBID/.restart_cmd`, or if it purposefully writes nothing to the file. Non-zero values indicate that `erestart.`*`method_name`* was not able to restart the job.

For user-level checkpointing, `erestart.`*`method_name`* must collect the exit code from the job. Then, `erestart.`*`method_name`* must exit with the same exit code as the job. Otherwise, the job's exit status is not reported correctly to LSF. Kernel-level checkpointing works differently and does not need this information from `erestart.`*`method_name`* to restart the job.

erestart.*method_name*

- Must have access to the original command line. It is important the `erestart.`*`method_name`* have access to the original command line used to start the job.
- `erestart.`*`method_name`* must return, it should not run the application to restart the job.

Note: Any information echkpnt writes to `stderr` is considered by `sbatchd` as an `echkpnt` failure. However, not all errors are fatal. If the `chkpnt` explicitly writes to `stdout` or `stderr` "`Checkpoint done`", `sbatchd` assumes `echkpnt` has succeeded.

Configuring LSF to recognize the custom echkpnt and erestart

You can set the following parameters in `lsf.conf` or as environment variables. If set in `lsf.conf`, these parameters apply globally to the cluster and will be the default values. Parameters specified as environment variables override the parameters specified in `lsf.conf`.

If you set parameters in `lsf.conf`, reconfigure your cluster with `lsadmin reconfig` and `badmin mbdrestart` so that changes take effect.

1. Set LSB_ECHKPNT_METHOD=*method_name* in `lsf.conf` or as an environment variable

   OR

   When you submit the job, specify the checkpoint and restart method. For example:

   ```
   % bsub -k "mydir method=myapp" job1
   ```

2. Copy your `echkpnt.`*`method_name`* and `erestart.`*`method_name`* to LSF_SERVERDIR.

   OR

   If you want to specify a different directory than LSF_SERVERDIR, in `lsf.conf` or as an environment variable set LSB_ECHKPNT_METHOD_DIR= absolute path to the directory in which your `echkpnt.`*`method_name`* and `erestart.`*`method_name`* are located.

   The checkpoint method directory should be accessible by all users who need to run the custom `echkpnt` and `erestart` programs.

3. (Optional)

   To save standard error and standard output messages for `echkpnt.`*`method_name`* and `erestart.`*`method_name`* set LSB_ECHKPNT_KEEP_OUTPUT=y in `lsf.conf` or as an environment variable.

   The `stdout` and `stderr` output generated by `echkpnt.` *`method_name`* will be redirected to:

   - `checkpoint_dir/$LSB_JOBID/echkpnt.out`
   - `checkpoint_dir/$LSB_JOBID/echkpnt.err`

   The `stdout` and `stderr` output generated by `erestart.`*`method_name`* will be redirected to:

   - `checkpoint_dir/$LSB_JOBID/erestart.out`
   - `checkpoint_dir/$LSB_JOBID/erestart.err`

   Otherwise, if LSB_ECHKPNT_KEEP_OUTPUT is not defined, standard error and output will be redirected to `/dev/null` and discarded.

Checkpointing a Job

Before LSF can checkpoint a job, it must be made checkpointable. LSF provides automatic and manual controls to make jobs checkpointable and to checkpoint jobs. When working with checkpointable jobs, a checkpoint directory must always be specified. Optionally, a checkpoint period can be specified to enable periodic checkpointing.

When a job is checkpointed, LSF performs the following actions:

1. Stops the job if its running
2. Creates the checkpoint file in the checkpoint directory
3. Restarts the job

Prerequisites

LSF can create a checkpoint for any eligible job. Review the discussion about "Approaches to Checkpointing" on page 309 to determine if your application and environment are suitable for checkpointing.

In this section

The Checkpoint Directory

A checkpoint directory must be specified for every checkpointable job and is used to store the files to restart a job. The directory must be writable by the job owner. To restart the job on another host (job migration), the directory must be accessible by both hosts. LSF does not delete the checkpoint files; checkpoint file maintenance is the user's responsibility.

LSF writes the checkpoint file in a directory named with the job ID of the job being checkpointed under the checkpoint directory. This allows LSF to checkpoint multiple jobs to the same checkpoint directory. For example, when you specify a checkpoint directory called `my_dir` and when job 123 is checkpointed, LSF will save the checkpoint file in:

```
my_dir/123/
```

When LSF restarts a checkpointed job, it renames the checkpoint directory using the job ID of the new job and creates a symbolic link from the old checkpoint directory to the new one. For example, if a job with job ID 123 is restarted with job ID 456 the checkpoint directory will be renamed to:

```
my_dir/456/
```

Making Jobs Checkpointable

Making a job checkpointable involves specifying the location of a checkpoint directory to LSF. This can be done manually on the command line or automatically through configuration.

Manually

Manually making a job checkpointable involves specifying the checkpoint directory on the command line. LSF will create the directory if it does not exist. A job can be made checkpointable at job submission or after submission.

At job submission

Use the `-k "checkpoint_dir"` option of `bsub` to specify the checkpoint directory for a job at submission. For example, to specify `my_dir` as the checkpoint directory for `my_job`:

```
% bsub -k "my_dir" my_job
Job <123> is submitted to default queue <default>.
```

After job submission

Use the `-k "checkpoint_dir"` option of `bmod` to specify the checkpoint directory for a job after submission. For example, to specify `my_dir` as the checkpoint directory for a job with job ID 123:

```
% bmod -k "my_dir" 123
Parameters of job <123> are being changed
```

Automatically

Automatically making a job checkpointable involves submitting the job to a queue that is configured for checkpointable jobs. To configure a queue, edit `lsb.queues` and specify the checkpoint directory for the CHKPNT parameter on a queue. The checkpoint directory must already exist, LSF will not create the directory.

For example, to configure a queue for checkpointable jobs using a directory named `my_dir`:

```
Begin Queue
  ...
  CHKPNT=my_dir
  DESCRIPTION = Make jobs checkpointable using "my_dir"
  ...
End Queue
```

Manually Checkpointing Jobs

LSF provides the `bchkpnt` command to manually checkpoint jobs. LSF can only perform a checkpoint for checkpointable jobs as described in "Making Jobs Checkpointable" on page 315. For example, to checkpoint a job with job ID 123:

```
% bchkpnt 123
Job <123> is being checkpointed
```

Interactive jobs (`bsub -I`) cannot be checkpointed.

Checkpointing and killing a job

By default, after a job has been successfully checkpointed, it continues to run. Use the `bchkpnt -k` command to kill the job after the checkpoint file has been created. Killing the job ensures the job does not do any processing or I/O after the checkpoint. For example, to checkpoint and kill a job with job ID 123:

```
% bchkpnt -k 123
Job <123> is being checkpointed
```

Enabling Periodic Checkpointing

Periodic checkpointing involves creating a checkpoint file at regular time intervals during the execution of your job. LSF provides the ability to enable periodic checkpointing manually on the command line and automatically through configuration. Automatic periodic checkpointing is discussed in "Automatically Checkpointing Jobs" on page 318. LSF can only perform a checkpoint for checkpointable jobs as described in "Making Jobs Checkpointable" on page 315.

Manually enabling periodic checkpointing involves specifying a checkpoint period in minutes.

At job submission

LSF uses the `-k "checkpoint_dir checkpoint_period"` option of `bsub` to enable periodic checkpointing at job submission. For example, to periodically checkpoint `my_job` every 2 hours (120 minutes):

```
% bsub -k "my_dir 120" my_job
Job <123> is submitted to default queue <default>.
```

After job submission

LSF uses the `-p` *`period`* option of `bchkpnt` to enable periodic checkpointing after submission. When a checkpoint period is specified after submission, LSF checkpoints the job immediately then checkpoints it again after the specified period of time. For example, to periodically checkpoint a job with job ID 123 every 2 hours (120 minutes):

```
% bchkpnt -p 120 123
Job <123> is being checkpointed
```

You can also use the `-p` option of `bchkpnt` to change a checkpoint period. For example, to change the checkpoint period of a job with job ID 123 to every 4 hours (240 minutes):

```
% bchkpnt -p 240 123
Job <123> is being checkpointed
```

Disabling periodic checkpointing

To disable periodic checkpointing, specify a period of 0 (zero). For example, to disable periodic checkpointing for a job with job ID 123:

```
% bchkpnt -p 0 123
Job <123> is being checkpointed
```

Automatically Checkpointing Jobs

Automatically checkpointing jobs involves submitting a job to a queue that is configured for periodic checkpointing. To configure a queue, edit `lsb.queues` and specify a checkpoint directory and a checkpoint period for the CHKPNT parameter for a queue. The checkpoint directory must already exist, LSF will not create the directory. The checkpoint period is specified in minutes. All jobs submitted to the queue will be automatically checkpointed. For example, to configure a queue to periodically checkpoint jobs every 4 hours (240 minutes) to a directory named `my_dir`:

```
Begin Queue
  ...
  CHKPNT=my_dir 240
  DESCRIPTION=Auto chkpnt every 4 hrs (240 min) to my_dir
  ...
End Queue
```

All jobs submitted to a queue configured for checkpointing can also be checkpointed using `bchkpnt`. Jobs submitted and modified using `-k`, `-r`, `-p`, and `-kn` options override queue configured options.

Restarting Checkpointed Jobs

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 is restarted, LSF performs the following actions:

1. LSF re-submits the job to its original queue as a new job and a new job ID is assigned
2. When a suitable host is available, the job is dispatched
3. The execution environment is recreated from the checkpoint file
4. The job is restarted from its last checkpoint.

This can be done manually from the command line, automatically through configuration, and when a job is migrated.

Requirements

LSF can restart a job from its last checkpoint on the execution host, or on another host if the job is migrated. To restart a job on another host, both hosts must:

- Be binary compatible
- Run the same dot version of the operating system. Unpredictable results may occur if both hosts are not running the exact same OS version.
- Have access to the executable
- Have access to all open files (LSF must locate them with an absolute path name)
- Have access to the checkpoint file

Manually restarting jobs

Use the `brestart` command to manually restart a checkpointed job. To restart a job from its last checkpoint, specify the checkpoint directory and the job ID of the checkpointed job. For example, to restart a checkpointed job with job ID 123 from checkpoint directory `my_dir`:

```
% brestart my_dir 123
Job <456> is submitted to default queue <default>
```

The `brestart` command allows you to change many of the original submission options. For example, to restart a checkpointed job with job ID 123 from checkpoint directory `my_dir` and have it start from a queue named `priority`:

```
% brestart -q priority my_dir 123
Job <456> is submitted to queue <priority>
```

Migrating Jobs

Migration is the process of moving a checkpointable or rerunnable job from one host to another host.

Checkpointing enables a migrating job to make progress by restarting it from its last checkpoint. Rerunnable non-checkpointable jobs are restarted from the beginning. LSF provides the ability to manually migrate jobs from the command line and automatically through configuration. When a job is migrated, LSF performs the following actions:

1. Stops the job if it is running
2. Checkpoints the job if it is checkpointable
3. Kills the job on the current host
4. Restarts or reruns the job on the next available host, bypassing all pending jobs

Requirements

To migrate a checkpointable job to another host, both hosts must:

- Be binary compatible
- Run the same dot version of the operating system. Unpredictable results may occur if both hosts are not running the exact same OS version.
- Have access to the executable
- Have access to all open files (LSF must locate them with an absolute path name)
- Have access to the checkpoint file

Manually migrating jobs

Use the `bmig` command to manually migrate jobs. Any checkpointable or rerunnable job can be migrated. 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

% bhist -l 123
Job Id <123>, User <user1>, Command <my_job>
Tue Feb 29 16:50:27: Submitted from host <hostA> to Queue <default>, C
  WD <$HOME/tmp>, Checkpoint directory <chkpnt_dir/123>;
Tue Feb 29 16:50:28: Started on <hostB>, Pid <4705>;
Tue Feb 29 16:53:42: Migration requested;
Tue Feb 29 16:54:03: Migration checkpoint initiated (actpid 4746);
Tue Feb 29 16:54:15: Migration checkpoint succeeded (actpid 4746);
Tue Feb 29 16:54:15: Pending: Migrating job is waiting for reschedule;
Tue Feb 29 16:55:16: Started on <hostC>, Pid <10354>.

Summary of time in seconds spent in various states by Tue Feb 29 16:57:26
PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
62       0        357      0        0        0        419
```

Automatically Migrating Jobs

Automatic job migration works on the premise that if a job is suspended (SSUSP) for an extended period of time, due to load conditions or any other reason, the execution host is heavily loaded. To allow the job to make progress and to reduce the load on the host, a migration threshold is configured. LSF allows migration thresholds to be configured for queues and hosts. The threshold is specified in minutes.

When configured on a queue, the threshold will apply to all jobs submitted to the queue. When defined at the host level, the threshold will apply to all jobs running on the host. When a migration threshold is configured on both a queue and host, the lower threshold value is used. If the migration threshold is configured to 0 (zero), the job will be migrated immediately upon suspension (SSUSP).

You can use `bmig` at anytime to override a configured threshold.

Configuring queue migration threshold

To configure a migration threshold for a queue, edit `lsb.queues` and specify a threshold for the MIG parameter. For example, to configure a queue to migrate suspended jobs after 30 minutes:

```
Begin Queue
  ...
  MIG=30        # Migration threshold set to 30 mins
  DESCRIPTION=Migrate suspended jobs after 30 mins
  ...
End Queue
```

Configuring host migration threshold

To configure a migration threshold for a host, edit `lsb.hosts` and specify a threshold for the MIG parameter for a host. For example, to configure a host to migrate suspended jobs after 30 minutes:

```
Begin Host
  HOST_NAME   r1m   pg   MIG # Keywords
  ...
  hostA       5.0   18   30
  ...
End Host
```

Requeuing migrating jobs

By default, LSF restarts or reruns a migrating job on the next available host, bypassing all pending jobs.

You can configure LSF to requeue migrating jobs rather than immediately restarting them. Jobs will be requeued in PEND state and ordered according to their original submission time and priority. To requeue migrating jobs, edit `lsf.conf` and set LSB_MIG2PEND=1.

Additionally, you can configure LSF to requeue migrating jobs to the bottom of the queue by editing `lsf.conf` and setting LSB_MIG2PEND=1 and LSB_REQUEUE_TO_BOTTOM=1.

C H A P T E R

24

Chunk Job Dispatch

Contents

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` 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 in normal queues can under-utilize 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.

Configuring a Chunk Job Dispatch

CHUNK_JOB_SIZE (lsb.queues)

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
```

After adding CHUNK_JOB_SIZE to `lsb.queues`, use `badmin reconfig` to reconfigure your cluster.

By default, CHUNK_JOB_SIZE is not enabled.

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_DURATION (lsb.params)

If CHUNK_JOB_DURATION is set in `lsb.params`, jobs submitted to a chunk job queue are only chunked if the job has a CPU limit or run limit set in the queue (CPULIMIT or RUNLMIT) or specified at job submission (`-c` or `-W` `bsub` options) that is less than or equal to the value of CHUNK_JOB_DURATION.

Jobs are not chunked if:

- CPU limit or a run limit is greater than the value of CHUNK_JOB_DURATION.
  or
- No CPU limit and no run limit are specified in the queue (CPULIMIT and RUNLIMIT) or at job submission (`-c` or `-W` `bsub` options).

The value of CHUNK_JOB_DURATION is displayed by `bparams -l`.

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 job queues

CHUNK_JOB_SIZE is ignored and jobs are not chunked for the following queues:

- Interactive (INTERACTIVE = ONLY parameter)
- CPU limit greater than 30 minutes (CPULIMIT parameter)

  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)

  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`)

Jobs submitted with the corresponding `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)

Submitting and Controlling Chunk Jobs

When a job is submitted to a queue 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

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 (`bstop`)	`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 (`bkill`)	`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)
	`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 redispatched 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 22, "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 23, "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.

Enforcing 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.

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.

See Chapter 26, "Runtime Resource Usage Limits" for more information.

C H A P T E R Ë

25 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

Creating a Job Array

A job array is created at job submission time using the `-J` option of `bsub`. 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`

  Used with `end` to specify 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]
```

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.

To make a change to the maximum job array value, set MAX_JOB_ARRAY_SIZE in `lsb.params` to any number up to 65534. The maximum number of jobs in a job array cannot exceed this value.

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 335
- "Passing Arguments on the Command Line" on page 336

Preparing 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. 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 at to provide the job ID of the job array.

Standard input

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
```

Standard output and error

Use the `-o` option of `bsub` and the %I and %J variables when your executable writes to standard output and error.

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
```

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 sero (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.

Whole array dependency

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, you would use the following command:

```
% bsub -w "done(123)" -J "myArray2[1-1000]" myJob
```

Partial array dependency

To make a job or job array dependent on an existing job array you would use one of the following dependency conditions.

Condition	Description
numrun(jobArrayJobId, op num)	Evaluate the number of jobs in RUN state
numpend(jobArrayJobId, op num)	Evaluate the number of jobs in PEND state
numdone(jobArrayJobId, op num)	Evaluate the number of jobs in DONE state
numexit(jobArrayJobId, op num)	Evaluate the number of jobs in EXIT state
numended(jobArrayJobId, op num)	Evaluate the number of jobs in DONE and EXIT state
numhold(jobArrayJobId, op num)	Evaluate the number of jobs in PSUSP state
numstart(jobArrayJobId, op num)	Evaluate the number of jobs in RUN and SSUSP and USUSP state

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.

Job array status

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    4    0     0     0     0
```

Individual job status

Current

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                  myArray[8]   Feb 29 12:34
123    user1  PEND  default   hostA                  myArray[9]   Feb 29 12:34
123    user1  PEND  default   hostA                  myArray[10]  Feb 29 12:34
```

Past

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    *ray[10]  375     0       24      0       0       0       399
```

Specific job status

Current 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
```

Past 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`.

Whole array

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
```

Individual jobs

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]"
```

Groups of jobs

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 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.

Requeuing jobs in DONE state

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.

Requeuing Jobs in EXIT state

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.

Requeuing all jobs in an array regardless of job state

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.

Requeuing RUN jobs to PSUSP state

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.

Requeuing jobs in RUN state

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 "arrayName[indexList]%jobLimit" myJob
```

where:

%jobLimit 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

A job array job slot limit can be set at the time of submission using `bsub`, or after submission using `bmod`.

At Submission For example, to set a job array job slot limit of 100 jobs for a job array of 1000 jobs:

```
% bsub -J "jobArrayName[1000]%100" myJob
```

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
```

Changing a job array job slot limit

Changing a job array job slot limit is the same as setting it 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
```

Viewing a job array job slot limit

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:
 NJOBS PEND DONE RUN EXIT SSUSP USUSP PSUSP
    10    9    0    1    0     0     0     0
```

P A R T

V

Controlling Job Execution

Contents

C H A P T E R

26

Runtime Resource Usage Limits

Contents

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 either at the queue level by your LSF administrator (`lsb.queues`) or 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 are *soft* limits. See `setrlimit(2)` man page for concepts of hard and soft limits.

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 16, "Resource Allocation Limits" for more information.

Summary of resource usage limits

Limit	Job syntax (bsub)	Queue syntax (lsb.queues)	Fomat/Units
Core file size limit	**-C** *core_limit*	`CORELIMIT=`*`limit`*	*integer* KB
CPU time limit	**-c** *cpu_limit*	`CPULIMIT=[`*`default`*`]` *`maximum`*	[*hours*:]*minutes*[/*host_name* | /*host_model*]
Data segment size limit	**-D** *data_limit*	`DATALIMIT=[`*`default`*`]` *`maximum`*	*integer* KB
File size limit	**-F** *file_limit*	`FILELIMIT=`*`limit`*	*integer* KB
Memory limit	**-M** *mem_limit*	`MEMLIMIT=[`*`default`*`]` *`maximum`*	*integer* KB
Process limit	**-p** *process_limit*	`PROCESSLIMIT=[`*`default`*`]` *`maximum`*	*integer* KB
Run time limit	**-W** *run_limit*	`RUNLIMIT=[default] maximum`	[*hours*:]*minutes*[/*host_name* | /*host_model*]
Stack segment size limit	**-S** *stack_limit*	`STACKLIMIT=`*`limit`*	*integer* KB
Virtual memory limit	**-v** *swap_limit*	`SWAPLIMIT=`*`limit`*	*integer* KB
Thread limit	**-T** *thread_limit*	`THREADLIMIT=[`*`default`*`]` *`maximum`*	*integer*

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 incorrect	The default is ignored and the maximum limit is enforced
Both default and maximum limits are specified, and the maximum is incorrect	The maximum is ignored and the resource has no maximum limit, only a default limit
Both default and maximum limits are incorrect	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.
```

Enforcing 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.

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.

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
- 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 incorrect:

```
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 incorrect:

```
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).

Specifying job-level resource usage limits

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)	Fomat/Units
`-C` *core_limit*	`CORELIMIT=`*limit*	*integer* KB

Sets a per-process (soft) core file size limit in KB for each process that belongs to this batch job. 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)	Fomat/Units
`-c` *cpu_limit*	`CPULIMIT=`[*default*] *maximum*	[*hours*`:`]*minutes*[`/`*host_name* | `/`*host_model*]

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 357 for more information.

Data segment size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Fomat/Units
-D *data_limit*	`DATALIMIT=[`*default*`]` *maximum*	*integer* KB

Sets a per-process (soft) data segment size limit in KB for each process that belongs to this batch job. An `sbrk()` or `malloc()` call to extend the data segment beyond the data limit returns an error. The default is no soft limit.

File size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Fomat/Units
-F *file_limit*	`FILELIMIT=`*limit*	*integer* 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.queues)	Fomat/Units
-M *mem_limit*	`MEMLIMIT=[`*default*`]` *maximum*	*integer* KB

Sets the memory limit, in KB.

If LSB_MEMLIMIT_ENFORCE or LSB_JOB_MEMLIMIT in `lsf.conf` are set to `y`, 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:

- Windows NT
- Sun Solaris 2.x

Process limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Fomat/Units
-p *process_limit*	`PROCESSLIMIT=[`*default*`]` *maximum*	*integer*

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)	Fomat/Units
-W *run_limit*	`RUNLIMIT=[default] maximum`	[*hours*:]*minutes*[/*host_name* | /*host_model*]

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 357 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 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)	Fomat/Units
-T *thread_limit*	`THREADLIMIT=[`*default*`]` *maximum*	*integer*

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)	Fomat/Units
`-S` *stack_limit*	`STACKLIMIT=`*`limit`*	*integer* KB

Sets a per-process (soft) stack segment size limit in KB for each process that belongs to this batch job. 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)	Fomat/Units
`-v` *swap_limit*	`SWAPLIMIT=`*`limit`*	*integer* KB

Sets the total process virtual memory limit to *swap_limit* in KB for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

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`, 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.

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`.

C H A P T E R

27

Load Thresholds

Contents

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.

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`.
- LSF does not suspend 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.
- LSF does not suspend a job 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.
- 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.

Specifying resume condition

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.

CHAPTER

28

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

Each batch job can be submitted with optional pre- and post-execution commands. Pre- and post-execution commands can be any excutable 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

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 both job-level and queue-level 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. Post-execution is only supported at the queue level.

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.

Job-level post-execution commands are not supported.

Queue-level commands

In some situations (for example, license checking), it is better to specify a queue-level pre-execution command instead of requiring every job be submitted with the `-E` option of `bsub`.

Queue-level commands run on the execution host before or after a job from the queue is run.

The LSF administrator uses the PRE_EXEC and POST_EXEC parameters in `lsb.queues` to set up queue-level pre- and post-execution commands.

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.

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. The post-processing of a repetitive job cannot be longer than the repetition period.

Configuring Pre- and Post-Execution Commands

Pre- and post-execution commands can be configured at the job level or on a per-queue basis.

Job-level commands

Job-level pre-execution commands require no configuration. Use the `bsub -E` option to specify an arbitrary command to run before the job starts.

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 pre-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.

  See "Automatic Job Requeue" on page 301 for more information.
- 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.

UNIX

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'
```

Windows

The pre- and post-execution commands are run under `cmd.exe /c`.

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.

See "LSB_PRE_POST_EXEC_USER parameter (lsf.sudoers)" on page 369.

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

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
```

LSB_PRE_POST_EXEC_USER parameter (lsf.sudoers)

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 pre- and post-execution commands.

Example

For 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 Reference* for information about the `lsf.sudoers` file.

C H A P T E R

29

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

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 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 job starters

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 378 for detailed information.

Queue-level job starters

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 376 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 28, "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.
- 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.
- Setting a job starter to `pvmjob` or `mpijob` allows you to run PVM or MPI jobs with LSF, where `pvmjob` and `mpijob` are job starters for parallel jobs written in PVM or MPI.

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 32, "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 376 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 `lsrun`, `lsgrun`, 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:

```
% lsrun "'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:\> lsrun 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 32, "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 374 for more information.

Configuring a queue-level job starter

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`*`] [%USRCMD] [`*`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 can also enclose the `%USRCMD` string in quotes or follow it with 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 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
- 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.

C H A P T E R

30

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

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 381

Modifying execution environments To modify the execution environment on the execution host, an `eexec` needs to be written. See "Working with eexec" on page 388

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 388

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

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	Arbitrary string format parameter containing the value of the `-a` option to `bsub` The value of `-a` is passed to `esub`, but it does not directly affect the other `bsub` parameters or behavior. LSB_SUB_ADDITIONAL cannot be changed in or added to LSB_SUB_MODIFY_FILE..
LSB_SUB_BEGIN_TIME	Begin time, in seconds since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_CHKPNT_DIR	Checkpoint directory
LSB_SUB_CLUSTER	Submission cluster name (MultiCluster only)
LSB_SUB_COMMAND_LINE	Job command
LSB_SUB_CHKPNT_PERIOD	Checkpoint period
LSB_SUB_DEPEND_COND	Dependency condition
LSB_SUB_ERR_FILE	Standard error file name

Option	Description
LSB_SUB_EXCEPTION	Exception condition
LSB_SUB_EXCLUSIVE	"Y" specifies exclusive execution
LSB_SUB_EXTSCHED_PARAM	Validate or modify `bsub -extsched` option
LSB_SUB_HOST_SPEC	Host specifier
LSB_SUB_HOSTS	List of execution host names
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 `bsub -wa`
LSB_SUB_JOB_WARNING_TIME_PERIOD	Job warning time period specified by `bsub -wt`
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	Always "SUB_RESET" if defined to indicate a `bmod` is being performed to reset the number of files to be transferred
LSB_SUB_OTHER_FILES_*number*	*number* is an index number indicating the particular file transfer value is the specified file transfer expression. For example, for `bsub -f "a > b" -f "c < d"`, 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
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
LSB_SUB_RES_REQ	Resource requirement string
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_RSS	Resident size limit
LSB_SUB_RLIMIT_RUN	Wall-clock run limit
LSB_SUB_RLIMIT_STACK	Stack size limit
LSB_SUB_RLIMIT_THREAD	Thread limit

Option	Description
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_SLA	SLA scheduling options

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.

LSB_SUB_ADDITIONAL cannot be changed in or added to LSB_SUB_MODIFY_FILE.

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
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 stderr to stdout 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
```

The bmod and brestart commands and esub

You can use the `bmod` command to modify job submission parameters, and `brestart` to restart checkpointed jobs. Like `bsub`, `bmod` and `brestart` also call `esub` if it exists. `bmod` and `brestart` cannot make changes to the job environment through `esub`. Environment changes only occur when `esub` is called by the original job submission with `bsub`.

How LSF supports multiple esub

LSF provides a master `esub` (`LSF_SERVERDIR/mesub`) to handle the invocation of individual esub executables and the job submission requirements of your applications. 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.

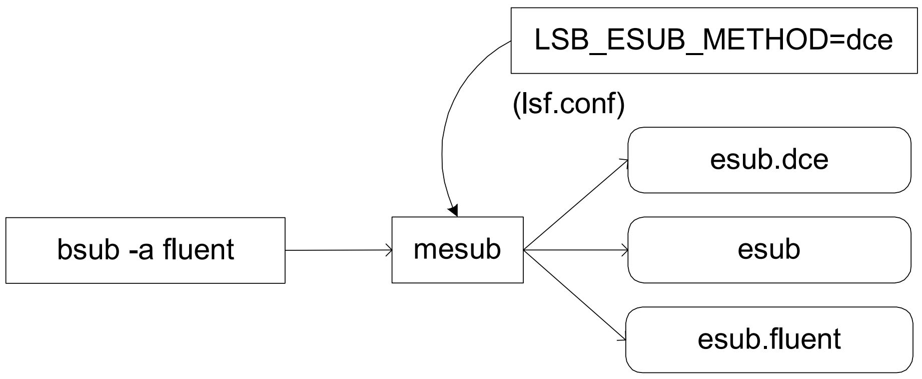
For example, `LSB_ESUB_METHOD="dce fluent"` defines DCE as the mandatory security system, and FLUENT as the mandatory application used on all jobs.

How master esub invokes application-specific esubs

`bsub` invokes `mesub` at job submission, which calls:

1. Mandatory `esub` programs defined by LSB_ESUB_METHOD
2. `esub` if it exists
3. application-specific `esub` programs if the `bsub -a` option is specified

Example



In this example, `esub.dce` is defined as a mandatory `esub`, an `esub` already exists in LSF_SERVERDIR, and the job is submitted as a FLUENT job to use `esub.fluent`.

Configuring 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.

Naming your esub

Use the following naming conventions:

- On UNIX, `LSF_SERVERDIR/esub.`*`application`*

  For example, `esub.fluent` for FLUENT jobs
- On Windows, `LSF_SERVERDIR\esub.`*`application`*`.[exe |bat]`

  For example, `esub.fluent.exe`

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).

esub.user is reserved

The name esub.user is reserved for backward compatibility. Do not use the name esub.user for your application-specific esub.

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 LS_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 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 LS_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 LS_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.

C H A P T E R

31

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

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 121 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:

- 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 Reference* for information about the `lsb.params` file.

LSF invokes the TERMINATE action when:

- The user or LSF administrator issues a `bkill` or `brequeue` 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 *Using the Platform LSF SDK* 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`.

JOB_CONTROLS parameter (lsb.queues)

The JOB_CONTROLS parameter has the following format:

```
Begin Queue
...
JOB_CONTROLS = SUSPEND[signal | CHKPNT | command] \
               RESUME[signal | command]   \
               TERMINATE[signal | CHKPNT | 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.

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

Using a command as a job control action

The following apply to a job control action that is a command:

- 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_JOBPIDS—a list of current process IDs of the job
- For the SUSPEND action command, the following environment variable is also set:

  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.
- 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 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_JOBPIDS;
     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 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 `bkill` command uses the TERMINATE action, so a deadlock results when the TERMINATE action itself contains the `bkill` command.

Any of the following job control specifications will cause a deadlock:

- JOB_CONTROLS=TERMINATE[bkill]
- 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 `lsrun` 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 `lsrun` (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.

P A R T

VI

Interactive Jobs

Contents

CHAPTER

32

Interactive Jobs with bsub

Contents

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 Reference* for information about configuring interactive queues in the `lsb.queues` file.

Interactive jobs with non-batch utilities

Non-batch utilities such as `lsrun`, `lsgrun`, 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 415.

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(1)` man page.

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.

Submitting an interactive job

Use the `bsub -I` option to submit an interactive batch job.

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`) or submitted to rerunnable queues (RERUNNABLE=y in `lsb.queues`).

Examples

- `% 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.

Submitting an interactive job by using a pseudo-terminal

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either `lsrun` or `bsub` LSF commands.

bsub -Ip To submit a batch interactive job by using a pseudo-terminal, use the `bsub -Ip` option.

When you specify the `-Ip` 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.

For example:

```
% bsub -Ip vi myfile
```

Submits a batch interactive job to edit `myfile`.

bsub -Is To submit a batch interactive job and create a pseudo-terminal with shell mode support, use the `bsub -Is` option.

When you specify the `-Is` 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`).

Example:

```
% bsub -Is csh
```

Submits a batch interactive job that starts up `csh` as an interactive shell.

Submitting an interactive job and redirect streams to files

bsub -i, -o, -e It is possible to 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.

For example:

```
% bsub -I -q interactive -e job.err lsmake
```

Saves the standard error stream in the `job.err` file, while standard input and standard output come from the terminal.

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 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.

For example:

```
% bsub -I myjob 2>mystderr 1>mystdout
```

In the above example, both `stderr` and `stdout` are written to `mystdout`.

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 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.

Configuring interactive batch job messaging

Messaging for interactive batch jobs can be specified cluster-wide or in the user environment.

Cluster level

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

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; >>

<<  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 `lsrun` 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>.
```

There are a few things to note:

- 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 378 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 only on SGI IRIX (6.4 and later).
- `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.

CHAPTER

33

Running Interactive and Remote Tasks

This chapter provides instructions for running tasks interactively and remotely with non-batch utilities such as `lsrun`, `lsgrun`, and `lslogin`.

Contents

Running Remote Tasks

`lsrun` is a non-batch utility to run tasks on a remote host. `lsgrun` 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 `lsgrun` options, see the `lsrun(1)` and `lsgrun(1)` man pages.

In this section

Running a task on the best available host

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.

Running 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`.

For example:

```
% 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 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.

Running a task on a specific host

If you want to run your task on a particular host, use the `lsrun -m` option:

```
% lsrun -m hostD mytask
```

Running 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.

The `-P` option of `lsrun` specifies that the job should be run using a pseudo-terminal:

```
% lsrun -P vi
```

Running 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.

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
```

Running parallel tasks

lsgrun -p

The `-p` option tells `lsgrun` that the task specified should be run in parallel. See `lsgrun(1)` for more details.

To remove the `/tmp/core` file from all 3 hosts, enter:

```
% lsgrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

Running tasks on hosts specified by a file

lsgrun -f *host_file* 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

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 `lsrcp` 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.

Redirecting 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.

For example, 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 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.

Logging on to the least loaded host

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.

Logging on to a host with specific resources

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

Starting an xterm

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`.

Setting 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:

   ```
   lsrun -R "type==any order[cpu:mem:login]" lsbg xterm -sb -
   ls -display your_PC:0.0
   ```
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.

Starting an xterm in Exceed

To start an `xterm`:

- Double-click the **Best** icon.

  You will get an `xterm` started on the least loaded host in the cluster and 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" lsbg 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]" lsbg $@
exit $?
```

Which would simplify the command string to:

```
lslaunch xterm -sb -ls -display your_PC:0.0
```

Taking this one step further, you could have `lsxterm`:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" lsbg xterm -sb -sl $@
exit $?
```

Which would simplify the command string to:

```
lsxterm -display your_PC:0.0
```

PART

VII

Running Parallel Jobs

Contents

C H A P T E R

34

Running Parallel Jobs

Contents

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.

For information about writing parallel applications for use with the Platform Parallel product, see *Using Platform Parallel (OBSOLETE)*.

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.

LSF includes example shell scripts for running PVM (`pvmjob`), P4 (`p4job`), and MPI (`mpijob`) programs as parallel batch jobs. These scripts are installed in the LSF_BINDIR directory as defined in the `lsf.conf` file.

You can modify these scripts to support more parallel packages.

For more information, see:

- "Submitting Parallel Jobs" on page 432

Using 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 376 for more information about job starters.

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.

- “Specifying the number of processors” on page 432
- “Submitting PVM Jobs to LSF” on page 433
- “Submitting MPI Jobs” on page 434

Specifying the number of processors

When submitting a parallel job that requires multiple processors, you can specify the exact number of processors to use.

To submit a parallel job, use `bsub -n` and specify multiple processors.

Example

```
% bsub -n 4 myjob
```

This command submits myjob as a parallel job. The job is started when 4 job slots are available.

Submitting PVM Jobs to LSF

Parallel Virtual Machine (PVM) is a parallel programming system distributed by Oak Ridge National Laboratory. PVM programs are controlled by the PVM hosts file, which contains host names and other information.

pvmjob script

The `pvmjob` shell script supplied with LSF can be used to run PVM programs as parallel LSF jobs. The `pvmjob` script reads the LSF environment variables, sets up the PVM hosts file and then runs the PVM job. If your PVM job needs special options in the hosts file, you can modify the `pvmjob` script.

Example

For example, if the command line to run your PVM job is:

```
% myjob data1 -o out1
```

the following command submits this job to LSF to run on 10 hosts:

```
% bsub -n 10 pvmjob myjob data1 -o out1
```

Other parallel programming packages can be supported in the same way. The `p4job` shell script runs jobs that use the P4 parallel programming library. Other packages can be handled by creating similar scripts.

Submitting MPI Jobs

The Message Passing Interface (MPI) is a portable library that supports parallel programming. LSF supports MPICH, a joint implementation of MPI by Argonne National Laboratory and Mississippi State University. This version supports both TCP/IP and IBM's Message Passing Library (MPL) communication protocols.

mpijob script

LSF provides an `mpijob` shell script that you can use to submit MPI jobs to LSF. The `mpijob` script writes the hosts allocated to the job by LSF to a file and supplies the file as an option to MPICH's `mpirun` command.

mpijob syntax

`mpijob -tcp mpirun` *program arguments*

Write the LSF hosts to a `PROCGROUP` file, supply the `-p4pg` *`procgroup_file`* option to the `mpirun` command, and use the TCP/IP protocol. This is the default.

`mpijob -mpl mpirun` *program arguments*

Write the LSF hosts to a `MACHINE` file, supply the `-machinefile` *`machine_file`* option to the `mpirun` command, and use the MPL on an SP-2 system.

- *program*—The parallel executable to be run
- *arguments*—Any arguments required by the parallel executable

Example

To submit a job requesting four hosts and using the default TCP/IP protocol, use:

```
% bsub -n 4 mpijob mpirun myjob
```

Submitting jobs to a pool of IBM SP-2 nodes

Before you can submit a job to a particular pool of IBM SP-2 nodes, an LSF administrator must install the SP-2 ELIM. The SP-2 ELIM provides the pool number and lock status of each node.

To submit the same job to run on four nodes in pool 1 on an IBM SP-2 system using MPL, use:

```
% bsub -n 4 -R "pool == 1" mpijob -mpl mpirun myjob
```

To submit the same job to run on four nodes in pool 1 that are not locked (dedicated to using the High Performance Switch) on an SP-2 system using MPL, use:

```
% bsub -n 4 -q mpiq -R "pool == 1 && lock == 0" mpijob -mpl mpirun myjob
```

Submitting jobs using the IBM SP-2 High Performance switch

Before you can submit a job using the IBM SP-2 High Performance Switch in dedicated mode, an LSF administrator must set up a queue for automatic requeue on job failure. The job queue will automatically requeue a job that failed because an SP-2 node was locked after LSF selected the node but before the job was dispatched.

Note that exclusive job requeue does not work for parallel jobs.

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.

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 `lsrun` in the background to start each component.

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 to which the job is submitted, or more processors than the maximum PROCLIMIT cannot use the queue and 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.

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 Mandatory First Execution Host

In general, the first execution host satisfies certain resource requirements that might not be present on other available hosts.

LSF normally selects the first execution host dynamically according to the resource availability and host load for a parallel job. You can also specify a *mandatory* first execution host.

Specify a mandatory first execution host

Use an exclamation point (!) to indicate mandatory first execution host. You can specify first execution host at job submission, or in the queue definition.

Job level Use the `-m` option of `bsub`:

```
% bsub -n 32 -m "hostA! hostB hostC" myjob
```

`hostA` is the mandatory first execution host.

Queue level Specify the first execution host in the list of hosts in the HOSTS parameter in lsb.queues:

```
HOSTS = hostA! hostB hostC
```

The queue-level specification of mandatory first execution host applies to all jobs submitted to the queue.

Rules The following rules apply when specifying first execution host:

- First execution host cannot be a host group or host partition, even if only one host is in the group or partition. Jobs that specify a host group or host partition as first execution host are rejected.
- The first execution host must satisfy the corresponding resource requirement specified at submission or on the queue. The order string in the resource requirement is ignored: the select, span and rusage strings in a resource requirement apply to the first execution host but the order string does not.

  If the specified first execution host does not satisfy the resource requirement, the job will stay in the pending state until the resource requirement is satisfied.
- The first execution host can appear anywhere in the host string, but you can specify only one first execution host.

  If multiple first execution hosts are specified in HOSTS on a queue, only the first valid first execution host is used, the others will be ignored as if they were not specified.
- The keyword `all` is ignored if it is specified in the execution host list. The keyword `others` is valid in the host list, but it cannot be the first execution host. Both keywords are ignored if they are specified as the first execution host.

If the first execution host is incorrect at job submission, the job will be rejected. If incorrect configurations exist on the queue level, warning messages will be logged and displayed when LSF starts, restarts or is reconfigured.

Job chunking

Specifying a mandatory first execution host affects job chunking. For example, the following jobs have different job requirements, and will not be placed in the same job chunk:

```
% bsub -m "hostA! hostB hostC" myjob
% bsub -m "hostA hostB hostC" myjob
```

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`).

Syntax

Two kinds of `span` string are supported:

- **`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 (*value*) on each host that should be allocated to the job.

  where *value* is:

  - Default `ptile` value, specified by *n* processors. For example:

    `span[ptile=4]`

    LSF allocates 4 processors on each available host, regardless of how many processors the host has.

  - Predefined `ptile` value, specified by ’!’. For example:

    `span[ptile='!']`

    uses the predefined maximum job slot limit in `lsb.hosts` (MXJ per host type/model) as its value.

    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.

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:

- 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.
- 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.
- You cannot mix host model and host type in the same `span` string. The following `span` strings are incorrect:

  ```
  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.
- 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. The following `span` strings are valid:

  ```
  same[type:model] span[LINUX:2,SGI:4]
  ```

  LINUX and SGI are both host types and can appear in the same `span` string.

  ```
  same[type:model] span[PC233:2,PC1133:4]
  ```

  `PC233` and `PC1133` are both host models and can appear in the same `span` string.

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.

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`

Using LSF Make to Run Parallel Jobs

For parallel jobs that have a variety of different components to run, you can use Platform Make. Create a makefile that lists all the components of your batch job and then submit the Platform Make command to LSF.

Example

The following example shows a `bsub` command and makefile for a simple parallel job.

```
% bsub -n 4 lsmake -f Parjob.makefile
Job <3858> is submitted to default queue <normal>.
```

Parjob.makefile

```
# Makefile to run example parallel job using lsbatch and
# Platform Make

all: part1 part2 part3 part4

part1 part2 part3: myjob data.$@

part4: myjob2 data.part1 data.part2 data.part3
```

The batch job has four components. The first three components run the `myjob` command on the `data.part1`, `data.part2` and `data.part3` files. The fourth component runs the `myjob2` command on all three data files. There are no dependencies between the components, so Platform Make runs them in parallel.

Limiting the Number of Processors Allocated

Use the PROCLIMIT parameter in `lsb.queues` to limit the number of processors that can be allocated to a parallel job in the queue.

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 <= *minimum* <= *default* <= *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.

Jobs that specify fewer processors than the minimum PROCLIMIT or more processors than the maximum PROCLIMIT cannot use this queue and are rejected.

If a default value for PROCLIMIT is specified in the queue, jobs submitted without specifying `-n` use the default number of processors. If the queue 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 the queue has no PROCLIMIT, 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 for the queue, it remains in PEND state, and the pending reason changes to the following:

```
Job no longer satisfies queue 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.

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
`% bsub -n 2 myjob`	The job `myjob` runs on 2 processors.
`% bsub -n 4 myjob`	The job `myjob` is rejected from the queue because it requires more than the maximum number of processors configured for the queue (3).
`% bsub -n 2,3 myjob`	The job `myjob` runs on 2 or 3 processors.
`% bsub -n 2,5 myjob`	The job `myjob` runs on 2 or 3 processors, depending on how many slots are currently available on the host.
`% bsub myjob`	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
`% bsub -n 5 myjob`	The job `myjob` runs on 5 processors.
`% bsub -n 2 myjob`	The job `myjob` is rejected from the queue because the number of processors requested is less than the minimum number of processors configured for the queue (3).
`% bsub -n 4,5 myjob`	The job `myjob` runs on 4 or 5 processors.
`% bsub -n 2,6 myjob`	The job `myjob` runs on 3 to 6 processors.
`% bsub -n 4,9 myjob`	The job `myjob` runs on 4 to 8 processors.
`% bsub myjob`	The default number of processors is equal to the minimum number (3). The job `myjob` 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
`% bsub myjob`	Because a default number of processors is configured, the job `myjob` runs on 6 processors.

Reserving Processors

About processor reservation

When parallel jobs have to compete with sequential jobs for resources, job 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 resources 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 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.

Configuring processor reservation

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 `rusage` 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 `rusage` 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 272.

lsb.queues 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
```

Enabling 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.

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.

Allowing Jobs to Use Reserved Job Slots

About backfill scheduling

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. 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

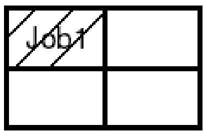
For backfill scheduling, LSF assumes that a job will run until its run limit expires. Backfill scheduling works most efficiently when all the jobs in the cluster have a run limit.

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.

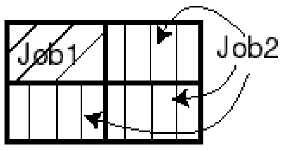
Once the big parallel job has reserved sufficient job slots, LSF calculates the start time of the big job, based on the run limits of the jobs currently running in the reserved slots. LSF cannot backfill if the big job is waiting for a job that has no run limit defined.

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.

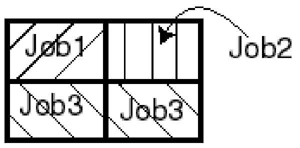
Example



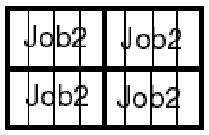
(a) Job1 started at 8:00 am.
Will finish at 10:00 am.



(b) Job2, submitted but can't start
since it needs 4 processors.
Remaining 3 reserved by Job2.



(c) At 8:30 am Job3 submitted.
Job3 backfills Job2.



(d) At 10:00 am, Job2 starts.

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.
- Jobs in a backfill queue cannot be preempted (a job in a backfill queue might be running in a reserved job slot, and starting a new job in that slot might delay the start of the big parallel job):
  - A backfill queue cannot be preemptable.
  - A preemptive queue whose priority is higher than the backfill queue cannot preempt the jobs in backfill queue.

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.

Configuring a backfill queue

To configure a backfill queue, define BACKFILL in `lsb.queues`.

Specify `Y` to enable backfilling. To disable backfilling, specify `N` or blank space.

Example

BACKFILL=Y

Enforcing run limits

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`).

Viewing information about job start time

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.

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:

dynamic priority = *number_shares* / (*cpu_time* * CPU_TIME_FACTOR + *run_time* * *number_CPUs* * RUN_TIME_FACTOR + (1 + *job_slots*)* RUN_JOB_FACTOR)

where *number_CPUs* is the number of CPUs used by the job.

Configuring parallel fairshare

To configure parallel fairshare:

1. Configure fairshare at the queue or host partition level as indicated in “Fairshare Scheduling” on page 201.
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 “Deadline Constraint Scheduling” on page 174. 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:

(deadline time - current time) > (CPU limit on candidate host / 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.

Configuring optimized preemption

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.

C H A P T E R

35

Advance Reservation

Contents

About Advance Reservation

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.

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.

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.

Configuring Advance Reservation

Advance reservation plugin

To enable advance reservation in your cluster, configure the advance reservation scheduling plugin `schmod_advrsv` in `lsb.modules`.

Configuring 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`.

Allowing users to create advance reservations

Advance reservation user policies

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, use the ResourceReservation section of `lsb.resources` to configure advance reservation policies for users.

ResourceReservation section (lsb.resources)

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`.

Examples

- 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.:

```
% user1@hostB> brsvadd -m "hostA" -n 1 -u "user2" -t "5:9:0-5:11:0"
Reservation "user2#2" is created
```

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
```

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 6.0. 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:

brsvadd	Add a reservation
brsvdel	Delete a reservation
brsvs	View reservations

Adding and removing reservations

By default, only LSF administrators or root can add or delete advance reservations.

brsvadd command

Use `brsvadd` to create new advance reservations. You must specify the following for the reservation:

- Number of processors to reserve

  This number should 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.

Specifying hosts for the reservation

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 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.

Adding a one-time reservation (-b and -e)

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.

Examples

- 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
  ```

Adding a recurring reservation (-t)

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.

Examples

- 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.

- The following command creates an advance reservation for 1024 processors on hosts `hostA` and `hostB` with more that 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
```

Removing an advance reservation (brsvdel)

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 only delete one reservation at a time.

For more information

See Chapter 10, "Time Syntax and Configuration" for more information about specifying time windows in LSF.

Viewing reservations

brsvs command

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 *`

Showing a weekly planner (brsvs -p)

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. MAX indicates the configured maximum number of job slots for the host.

```
% 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  (MAX = 1024)
Week: 11/11/2004 - 11/17/2004
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
...
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  (MAX = 1024)
Week: 11/11/2004 - 11/17/2004
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
```

bjobs command

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>;
```

Submitting and modifying jobs using advance reservations

Submitting and running jobs (bsub -U)

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.)

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.

Modifying jobs (bmod -U)

Administrators can use the `-U` option of `bmod` to change a job to another reservation ID. For example:

```
%bmod -U user1#0 1234
```

To cancel the reservation, use the `-Un` option of `bmod`. For example:

```
%bmod -Un 1234
```

Changing job termination time (bmod -t)

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
```

You must set LSB_MOD_ALL_JOBS=Y in `lsf.conf` to use `bmod -t` for advance reservations.

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 6.0 or later.

See the *Using Platform LSF MultiCluster* for more information.

Viewing historical accounting information for advance reservations

bacct -U 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`

Example

```
% 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 secon
```

P A R T

VIII

Monitoring Your Cluster

Contents

C H A P T E R

36

Event Generation

Contents

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 NT, use the Windows NT 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.

Custom event handling 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 479 and "Events list" on page 479 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 on 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.

C H A P T E R

37

Tuning the Cluster

Contents

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

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 359 for more details.

In this section

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 144.

Comparing LIM load thresholds

To tune LIM load thresholds, compare the output of `lsload` to the thresholds reported by `lshosts -l`.

The `lsload` and `lsmon` commands display an asterisk `*` next to each load index that exceeds its threshold.

Example For 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
...
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:

- `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 144 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

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.

By setting this parameter, you can reduce the time it takes to reconfigure a cluster and requests made to the file server. Only hosts listed in LSF_MASTER_LIST will read `lsf.shared` and `lsf.cluster.`*`cluster_name`*. Configuration information will then be propagated from the master LIM to slave-only LIMs.

Setting 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 with the commands `lsadmin reconfig` and `badmin 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 `badmin 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 need to restart all the LIMs in the cluster. Use the command `lsadmin reconfig`.

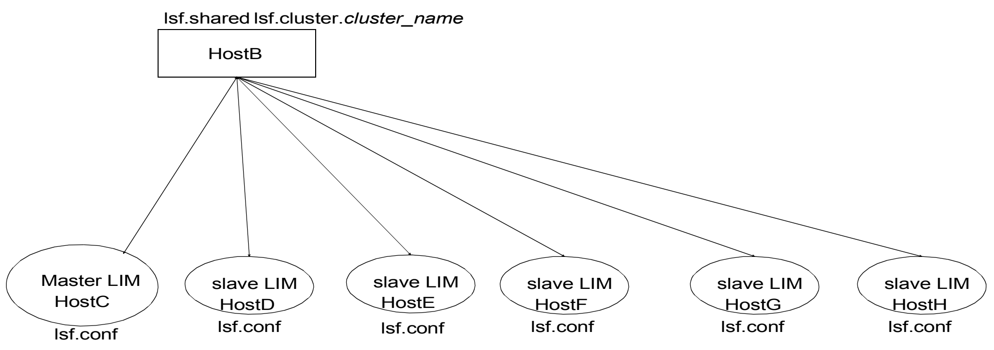
How LSF works with LSF_MASTER_LIST

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.



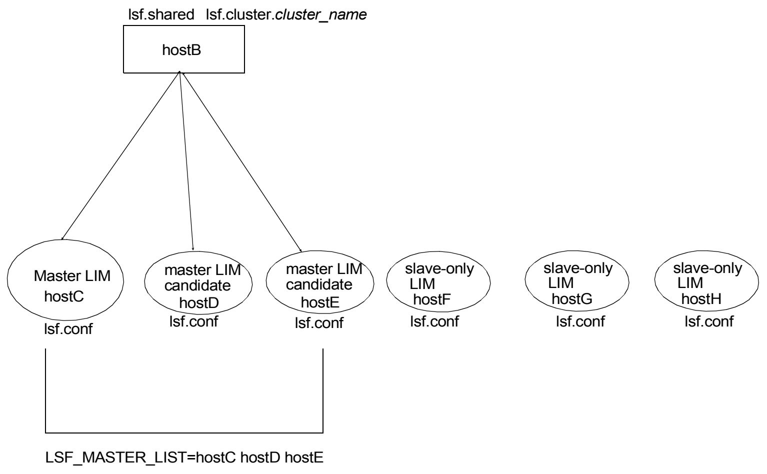
LSF_MASTER_LIST defined

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 undefined

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 490. 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.

Tuning mbatchd on UNIX

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.

Operating system support

See the Online Support area of the Platform Computing Web site at `www.platform.com` for the latest information about operating systems that support multithreaded `mbatchd`.

In this section

- "How mbatchd works without multithreading" on page 491
- "How mbatchd works with multithreading" on page 491
- "Setting a query-dedicated port for mbatchd" on page 492
- "Specifying an expiry time for child mbatchds" on page 492

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.

How mbatchd works with multithreading

To change `mbatchd` behavior to use multithreading, complete the following procedures:

1. Mandatory.

   Specify a query-dedicated port for the `mbatchd`. You do this by setting the parameter LSB_QUERY_PORT in `lsf.conf`.

   See "Setting a query-dedicated port for mbatchd" on page 492.
2. Optional.

   Set an interval of time to indicate when a new child `mbatchd` is to be forked. You do this by setting the parameter MBD_REFRESH_TIME in `lsb.params`. The default value of MBD_REFRESH_TIME is 5 seconds, and valid values are 5-300 seconds.

   See "Specifying an expiry time for child mbatchds" on page 492.

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.

- If MBD_REFRESH_TIME is < 10 seconds, 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 > 10 seconds, the child `mbatchd` exits at 10 seconds even if the job changes status or a new job is submitted before the 10 seconds
- If MBD_REFRESH_TIME > 10 seconds and no job changes status or a new job is submitted, the child `mbatchd` exits at MBD_REFRESH_TIME

Setting a query-dedicated port for mbatchd

To enable a query-dedicated port for `mbatchd` and change the default `mbatchd` behavior so that `mbatchd` forks a child `mbatchd` that can create threads, specify a port number with the parameter LSB_QUERY_PORT in `lsf.conf`.

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.

Where to go next If you want to change the default value for MBD_REFRESH_TIME (default: 5 seconds), proceed to "Specifying an expiry time for child mbatchds" on page 492.

Otherwise, you have completed configuration. Reconfigure the cluster with the `badmin mbdrestart` command.

Specifying an expiry time for child mbatchds

You define how often `mbatchd` forks a new child `mbatchd` with the parameter MBD_REFRESH_TIME in `lsb.params`.

The default value for this parameter is 5 seconds. Valid values are 5 to 300 seconds.

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`.
4. Save the `lsb.params` file.
5. Reconfigure the cluster:

```
% badmin reconfig
```

CHAPTER 38

Authentication

Controlling access to remote execution has two requirements:

- Authenticate the user.

  When a user executes a remote command, the command must be run with that user's permission. The LSF daemons need to know which user is requesting the remote execution.
- Check access controls on the remote host.

  The user must be authorized to execute commands remotely on the host.

This chapter describes user, host, and daemon authentication in LSF.

Contents

About User Authentication

LSF recognizes UNIX and Windows authentication environments, including different Windows domains and individual Windows workgroup hosts.

- In a UNIX environment, user accounts are validated at the system level, so your user account is valid on all hosts.
- In a Windows domain environment, user accounts are validated at the domain level, and your user account is valid on all hosts in your domain (and might be valid in other domains, if there is a trust relationship).
- In a Windows workgroup environment, each host authenticates the user account, so your local account is only valid on one host.

User authentication options

To enable LSF users to execute commands remotely, you must specify the authentication method LSF uses to authorize remote execution across the network.

You have the following choices:

- External authentication (`eauth`)
- Privileged ports (`setuid`)
- Identification daemon (`identd`)

If you change the authentication type while the LSF daemons are running, you must shut down and restart the LSF daemons on each LSF server host, so that the daemons will use the new authentication method.

External authentication (eauth)

External authentication uses the LSF `eauth` executable installed in LSF_SERVERDIR. Optionally, you may choose to write your own `eauth` executable that uses some site-specific authentication method such as Kerberos or DCE client authentication using the GSSAPI.

Examples of these can be found in the `LSF_MISC/examples/krb` and `LSF_MISC/examples/dce` directories. Installation instructions are found in the README file in these directories.

By default, `eauth` uses an internal key to encrypt authentication data. To use an external key to improve security, configure the parameter LSF_EAUTH_KEY in the `lsf.sudoers` file. The default `eauth` program is installed without `setuid` permission. If you use LSF_EAUTH_KEY, `eauth` must be `setuid`.

The `eauth` mechanism can pass data (such as authentication credentials) from users to execution hosts. The environment variable LSF_EAUTH_AUX_DATA specifies the full path to a file where data, such as a credential, is stored. The mechanisms of `eauth -c` and `eauth -s` allow the LSF daemons to pass this data using a secure exchange.

LSF_EAUTH in lsf.conf

Installation with lsfinstall sets LSF_AUTH=eauth in lsf.conf automatically. To use another authentication mechanism, you must change the value of LSF_AUTH and restart all LSF daemons.

eauth -c *host_name* When a command is invoked, the client program automatically executes `eauth -c host_name` to get the external authentication data, where *host_name* is the name of the host running the LSF daemon (for example, RES) on which the operation will take place. The external user authentication data is passed to LSF through the standard output of the `eauth` program.

eauth -s When the LSF daemon receives the request, it executes `eauth -s` under the primary LSF administrator user ID to process the user authentication data.

If your site cannot run authentication under the primary LSF administrator user ID, configure the parameter LSF_EAUTH_USER in the `/etc/lsf.sudoers` file.

The LSF daemon expects `eauth -s` to write to standard output:

- 1 if authentication succeeds
- 0 if authentication fails

The same `eauth -s` process can service multiple authentication requests; if the process terminates, the LSF daemon will re-invoke `eauth -s` on the next authentication request.

See the *Platform LSF Reference* for information about configuring the `lsf.sudoers` file.

Standard input stream for the eauth program

User authentication data is passed to `eauth -s` via its standard input. The standard input stream to `eauth` has the following format:

uid gid user_name client_addr client_port user_auth_data_len user_auth_data

where:

- *uid* is the user ID in ASCII of the client user
- *gid* is the group ID in ASCII of the client user
- *user_name* is the user name of the client user
- *client_addr* is the host address of the client host in ASCII dot notation
- *client_port* is the port number from where the client request is made
- *user_auth_data_len* is the length of the external authentication data in ASCII passed from the client
- *user_auth_data* is the external user authentication data passed from the client

Privileged ports authentication (setuid)

This is the mechanism most UNIX remote utilities use. The LSF commands must be installed as `setuid` programs and owned by root.

If a load-sharing program is owned by root and has the `setuid` bit set, the LSF API functions use a privileged port to communicate with LSF servers, and the servers accept the user ID supplied by the caller. This is the same user authentication mechanism as used by the UNIX `rlogin` and `rsh` commands.

When a `setuid` application calls the LSLIB initialization routine, a number of privileged ports are allocated for remote connections to LSF servers. The effective user ID then reverts to the real user ID. Therefore, the number of remote connections is limited.

An LSF utility reuses the connection to RES for all remote task executions on that host, so the number of privileged ports is only a limitation on the number of remote hosts that can be used by a single application, not on the number of remote tasks. Programs using LSLIB can specify the number of privileged ports to be created at initialization time.

LSF_EAUTH in lsf.conf

If you do not define LSF_AUTH in lsf.conf, privileged ports (setuid) authentication is the default user authentication used by LSF. Installation with lsfinstall sets LSF_AUTH=eauth automatically. To use setuid authentication, you must remove LSF_AUTH from lsf.conf.

Identification daemon (identd)

LSF also supports authentication using the RFC 931 or RFC 1413 identification protocols. Under these protocols, user commands do not need to be installed as `setuid` programs owned by root. You must install the `identd` daemon available in the public domain.

The RFC 1413 and RFC 931 protocols use an identification daemon running on each client host. RFC 1413 is a more recent standard than RFC 931. LSF is compatible with both. Using an identification daemon incurs more overhead, but removes the need for LSF applications to allocate privileged ports.

You should use identification daemons if your site cannot install programs owned by root with the `setuid` bit set, or if you have software developers creating new load-sharing applications in C using LSLIB.

An implementation of RFC 931 or RFC 1413 such as `pidentd` or `authd` can be obtained from the public domain. If you have Internet FTP access, a good source for identification daemons is host `ftp.lysator.liu.se`, directory `pub/ident/servers`.

LSF_EAUTH in lsf.conf

Installation with lsfinstall sets LSF_AUTH=eauth in lsf.conf automatically. To use identd authentication, you must set LSF_AUTH=ident in lsf.conf.

How LSF determines the user authentication method

LSF uses the LSF_AUTH parameter in the `lsf.conf` file to determine which type of authentication to use:

If LSF_AUTH is ...	LSF uses ...
`eauth`	External authentication (eauth)
Not defined	Privileged ports (setuid)
`ident`	Identification daemon (identd)

LSF_AUTH=eauth

`LSF_AUTH=eauth` is set automatically during installation with `lsfinstall`. LSF runs the external executable `eauth` in the LSF_SERVERDIR directory to perform the authentication.

If a load-sharing application is not `setuid` to root, library functions use a non-privileged port. If the LSF_AUTH parameter is not set in `lsf.conf`, the connection is rejected.

LSF_AUTH=ident or undefined

If LSF_AUTH is defined to be `ident`, RES on the remote host, or `mbatchd` in the case of a `bsub` command, contacts the identification daemon on the local host to verify the user ID. The identification daemon looks directly into the kernel to make sure the network port number being used is attached to a program being run by the specified user.

LSF allows both the `setuid` and identification daemon methods to be in effect simultaneously. If the effective user ID of a load-sharing application is root, then a privileged port number is used in contacting RES. RES always accepts requests from a privileged port on a known host even if LSF_AUTH is defined to be `ident`. If the effective user ID of the application is not root, and the LSF_AUTH parameter is defined to be `ident`, then a normal port number is used and RES tries to contact the identification daemon to verify the user's identity.

setuid permission on LSF administration commands

The LSF administration commands (`lsadmin` and `badmin`, etc.) are installed `setuid` by default. All other LSF commands except the administration commands can be run without `setuid` permission if an identification daemon is used.

If your file server does not permit `setuid` permission, you should install LSF_BINDIR on a file system that does allow `setuid`.

Security of LSF authentication

All authentication methods supported by LSF depend on the security of the root account on all hosts in the cluster. If a user can get access to the root account, they can subvert any of the authentication methods. There are no known security holes that allow a non-root user to execute programs with another user's permission.

Some system adminstrators have particular concerns about security schemes involving RFC 1413 identification daemons. When a request is coming from an unknown host, there is no way to know whether the identification daemon on that host is correctly identifying the originating user.

LSF only accepts job execution requests that originate from hosts within the LSF cluster, so the identification daemon can be trusted.

The system environment variable LSF_ENVDIR is used by LSF to obtain the location of `lsf.conf`, which points to the LSF configuration files. Any user who can modify system environment variables can modify LSF_ENVDIR to point to their own configuration and start up programs under the `lsfadmin` account.

All external binaries invoked by the LSF daemons (such as `esub`, `eexec`, `elim`, `eauth`, and queue level pre- and post-execution commands) are run under the `lsfadmin` account.

UNIX

By default, external authentication is installed on UNIX. If you use the identification protocol (`identd`) for authentication, LSF uses a port in the UNIX privileged port range, so it is not possible for an ordinary user to start a hacked identification daemon on an LSF host.

On UNIX, this means that authentication is done using privileged ports and binaries that need to be authenticated (for example, `bsub`) are installed `setuid` to root.

Windows

By default, external authentication is installed on Windows. You may disable external authentication by disabling the LSF_AUTH parameter in the `lsf.conf` file.

On Windows, privileged ports authentication does not provide any security because Windows does not have the concept of `setuid` binaries and does not restrict which binaries can use privileged ports. A security risk exists in that a user can discover the format of LSF protocol messages and write a program that tries to communicate with an LSF server. The LSF default external authentication should be used where this security risk is a concern.

Only the parameters LSF_STARTUP_USERS and LSF_STARTUP_PATH are used on Windows. You should ensure that only authorized users modify the files under the %SYSTEMROOT% directory.

Once the LSF services on Windows are started, they will only accept requests from LSF cluster administrators. To allow other users to interact with the LSF services, you must set up the `lsf.sudoers` file under the directory specified by the %SYSTEMROOT% environment variable.

See the *Platform LSF Reference* for the format of the `lsf.sudoers` file.

Correcting user authentication errors

If LSF cannot verify the user's identity, the error message `User permission denied` is displayed by LSF commands.

This error has several possible causes:

- The LSF applications are not installed `setuid`.
- The NFS directory is mounted with the `nosuid` option.
- The identification daemon is not available on the local or submitting host.
- External authentication failed.
- You configured LSF to use `ruserok()` and the client host is not found in either the `/etc/hosts.equiv` or the `$HOME/.rhosts` file on the master or remote host.

Password problem notification on Windows

A Windows job may not be able to run because of a problem with the user's LSF password (entered and updated using `lspasswd`). If LSF does not recognize the password, the problem could be:

- The user never gave their Windows user account password to LSF (`lspasswd`).
- The user changed their password in Windows but did not update LSF (`lspasswd`).

If a job is in PEND state and LSF cannot run it because of a password problem, by default, LSF puts the job into USUSP and then notifies the user via email. The user can fix the problem, and then use `bresume` to release the job from USUSP.

About Host Authentication

When a batch job or a remote execution request is received, LSF first determines the user's identity. Once the user's identity is known, LSF decides whether it can trust the host from which the request comes from.

Trust LSF host

LSF normally allows remote execution by all users except root, from all hosts in the LSF cluster; LSF trusts all hosts that are configured into your cluster. The reason behind this is that by configuring an LSF cluster you are turning a network of machines into a single computer. Users must have valid accounts on all hosts. This allows any user to run a job with their own permission on any host in the cluster. Remote execution requests and batch job submissions are rejected if they come from a host not in the LSF cluster.

A site can configure an external executable to perform additional user or host authorization. By defining LSF_AUTH to be `eauth` in `lsf.conf`, the LSF daemon will invoke `eauth -s` when it receives a request that needs authentication and authorization. For example, `eauth` can check if the client user is on a list of authorized users or if a host has the necessary privilege to be trusted.

/etc/hosts.equiv (UNIX)

If the LSF_USE_HOSTEQUIV parameter is set in the `lsf.conf` file, LSF uses the same remote execution access control mechanism as the `rsh` command. When a job is run on a remote host, the user name and originating host are checked using the `ruserok`(3) function on the remote host.

The `ruserok`(3) function checks in the `/etc/hosts.equiv` file and the user's `$HOME/.rhosts` file to decide if the user has permission to execute jobs.

The name of the local host should be included in this list. RES calls `ruserok()` for connections from the local host. `mbatchd` calls `ruserok()` on the master host, so every LSF user must have a valid account and remote execution permission on the master host.

The disadvantage of using the `/etc/hosts.equiv` and `$HOME/.rhosts` files is that these files also grant permission to use the `rlogin` and `rsh` commands without giving a password. Such access is restricted by security policies at some sites.

For more information

See the `hosts.equiv`(5) and `ruserok`(3) man pages for details on the format of the files and the access checks performed.

About Daemon Authentication

Daemon authentication

By default, LSF calls the `eauth` program only for user authentication (authenticate LSF user requests to either RES or `mbatchd`).

LSF can also authenticate the following communications between daemons, using the same `eauth` program:

- `mbatchd` requests to `sbatchd`
- `sbatchd` updates to `mbatchd`
- PAM to `sbatchd` interactions
- `mbatchd` to `mbatchd` (in a Platform MultiCluster environment)

The `eauth` can use these environment variables to provide context:

- LSF_EAUTH_CLIENT - sender of the authentication request
- LSF_EAUTH_SERVER - receiver of the authentication request

Configuring daemon authentication

Set LSF_AUTH_DAEMONS in `lsf.conf`. For example,

```
LSF_AUTH_DAEMONS=1
```

LSF in Multiple Authentication Environments

In some environments, such as a UNIX system or a Windows domain, you can have one user account that works on all hosts. However, when you build an LSF cluster in a heterogeneous environment, you can have a different user account on each system, and each system does its own password authentication.

This means that LSF cannot always use the submission account to run a job, because the job will fail if the execution host cannot validate the password of the account you used on the submission host.

In an environment of multiple authentication systems, user mapping determines which account LSF uses when it runs your job. User mapping can be defined all of the following ways:

- For clusters containing Windows hosts, LSF default user mapping (LSF_USER_DOMAIN in `lsf.conf`) might be enabled. This should be configured only once, when you install and set up LSF.
- User mapping at the user level (`lsb.hosts`) is configurable by the user.
- User mapping at the system level (`lsb.users`) is configurable by the administrator.

User Account Mapping

LSF allows user account mapping across a non-uniform user name space.

By default, LSF assumes uniform user accounts throughout the cluster. This means that jobs will be executed on any host with exactly the same user ID and user login name.

The LSF administrator can disable user account mapping.

For information about account mapping between clusters in a MultiCluster environment, see the *Using Platform LSF MultiCluster*.

Configuring user-level account mapping (.lsfhosts)

1. Set up a hidden `.lsfhosts` file in your home directory that tells what accounts to use when you send jobs to remote hosts and which remote users are allowed to run jobs under your local account. This is similar to the `.rhosts` file used by `rcp`, `rlogin` and `rsh`.
2. Specify each line in the form:

   `host_name user_name` `[`**`send`**`|`**`recv`**`]`

   where `send` indicates that if you send a job to host *host_name*, then the account *user_name* should be used, and `recv` indicates that your local account is enabled to run jobs from user *user_name* on host *host_name*. If neither `send` nor `recv` are specified, your local account can both send jobs to and receive jobs from the account *user_name* on *host_name*. Lines beginning with '#' are ignored. A plus sign (+) in the *host_name* or *user_name* field indicates any LSF host or user respectively.
3. Set the permission on your `.lsfhosts` file to read/write only by the owner. Otherwise, your `.lsfhosts` file is silently ignored

MultiCluster

The cluster name can be substituted for *host_name* in a MultiCluster environment. For more information, see the *Using Platform LSF MultiCluster*.

Example 1

For example, assume that `hostB` and `hostA` in your cluster do not share the same user name/user ID space. You have an account `user1` on host `hostB` and an account ruser_1 on host `hostA`. You want to be able to submit jobs from `hostB` to run on `hostA`. Set up your `.lsfhosts` files as follows:

On `hostB`:

```
% cat ~user1/.lsfhosts
hostA ruser_1 send
```

On `hostA`:

```
% cat ~ruser_1/.lsfhosts
hostB user1 recv
```

Example 2 As another example, assume you have account `user1` on host `hostB` and want to use the `lsfguest` account when sending jobs to be run on host `hostA`. The `lsfguest` account is intended to be used by any user submitting jobs from any LSF host. Set up your `.lsfhosts` files as follows:

On `hostB`:

```
% cat ~user1/.lsfhosts
hostA lsfguest send
```

On `hostA`:

```
% cat ~lsfguest/.lsfhosts
+  + recv
```

C H A P T E R

39

Job Email, and Job File Spooling

Contents

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.

-o and -e Options The output file created by the `-o` option 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.

If you specify a `-o` *output_file* option and do not specify a `-e` *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`.

The output files specified by the `-o` and `-e` options are created on the execution host.

See "Remote File Access" on page 516 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 -o and -e. 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` 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 `-o` or `-e` options 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 `-o` option of `bsub`, LSB_MAILSIZE is not set.

Directory for job output

The `-o` and `-e` options 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.

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 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.

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.

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.
- 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 Reference* for more information about the `bsub` and `bmod` commands, the JOB_SPOOL_DIR parameter in `lsb.params`, and the LSF_TMPDIR environment variable.

C H A P T E R

40

Non-Shared File Systems

Contents

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 516 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 515 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.

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 518 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 specific 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.

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 (lsrcp)

The LSF remote file access mechanism (`bsub -f`) uses `lsrcp` to process the file transfer. The `lsrcp` command tries to connect to RES on the submission host to handle the file transfer.

See "Remote File Access" on page 516 for more information about using `bsub -f`.

Limitations to lsrcp

Because LSF client hosts do not run RES, jobs that are submitted from client hosts should only specify `bsub -f` if `rcp` is allowed. You must set up the permissions for `rcp` if account mapping is used.

File transfer using `lscrp` is not supported in the following contexts:

- If LSF account mapping is used; `lsrcp` fails when running under a different user account
- LSF client hosts do not run RES, so `lsrcp` cannot contact RES on the submission host

See "User Account Mapping" on page 503 for more information.

Workarounds

In these situations, use the following workarounds:

rcp on UNIX

If `lsrcp` 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 `lsrcp` with your own file transfer mechanism as long as it supports the same syntax as `lsrcp`. This might be done to take advantage of a faster interconnection network, or to overcome limitations with the existing `lsrcp`. `sbatchd` looks for the `lsrcp` executable in the `LSF_BINDIR` directory as specified in the `lsf.conf` file.

C H A P T E R

41

Error and Event Logging

Contents

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 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 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 log

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:

- `lim.log.`*host_name*
- `res.log.`*host_name*
- `pim.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 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.

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 partioning

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`.

Configuring 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.

CHAPTER

42

Troubleshooting and Error Messages

Contents

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

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

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 `lsload` 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.

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.
```

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.

RES does not start

Check the RES error log.

UNIX If the RES is unable to read the `lsf.conf` file and does not know where to write error messages, it logs errors into `syslog(3)`.

Windows If the RES is unable to read the `lsf.conf` file and does not know where to write error messages, it logs errors into `C:\temp`.

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.
```

Check the RES error log on the remote host; this usually contains a more detailed error message.

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
```

If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

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.

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.

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"
```

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

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 531.

sbatchd starts but mbatchd does not

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
```

Check whether services are registered properly. See "Registering Service Ports" on page 85 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

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 host type or model

1 Start the host.

2 Run `lsadmin limstartup` to start up the LIMs on the host. For example:

```
# lsadmin limstartup 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` and be able to run the `rsh` command across all hosts without entering a password. See "Prerequisites" on page 62 for more details.

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, you can leave it as is. When automatic detection of host type or model fails, the type or model is set to DEFAULT. LSF will work on the host. A DEFAULT model may be inefficient because of incorrect CPU factors. A DEFAULT type may cause binary incompatibility because a job from a DEFAULT host type can be migrated to another.

DEFAULT host type or model

Viewing DEFAULT host type or model

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  RESOURCE
S
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 host type

1 Run `lim -t` on the host whose type is DEFAULT:

```
% lim -t
```

```
Host Type              : sun4
Host Architecture      : SUNWUltra2_200_sparcv9
Matched Type           : DEFAULT
Matched Architecture   : SUNWUltra2_300_sparc
Matched Model          : Ultra2
CPU Factor             : 20.2
```

Note the value of `Host Type` and `Host Architecture`.

2 Edit `lsf.shared`.

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
sun4
...
```

3 Save changes to `lsf.shared`.

4 Run `lsadmin reconfig` to reconfigure LIM.

Fixing DEFAULT host model

1 Run the `lim -t` command on the host whose model is DEFAULT:

```
% lim -t
```

```
Host Type              : sun4
Host Architecture      : SUNWUltra2_200_sparcv9
Matched Type           : DEFAULT
Matched Architecture   : SUNWUltra2_300_sparc
Matched Model          : DEFAULT
CPU Factor             : 20.2
```

Note the value of `Host Architecture`.

2 Edit `lsf.shared`.

In the `HostModel` section, add the new model with architecture and CPU factor. 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 as part of the 127 line limit.

Use the architecture detected with `lim -t`. 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.

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 85 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.

`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 88 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 <clname>@<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
```

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 RESs always use `lsf.conf` to set the debug environment. Application RESs are the RESs 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 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`.
- **`% 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 543.

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`.

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 540.

For a detailed description of `lsadmin` and `badmin`, see the *Platform LSF Reference*.

P A R T

IX

LSF Utilities

Contents

C H A P T E R

43

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

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 549
- "Local and Remote Modes" on page 550
- "Automatic Remote Execution" on page 551

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 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 556 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, `lstcsh` 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 Reference* for information about task lists and `lsf.task` file.

If the command can be executed on a remote host, `lstcsh` 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.

`lstcsh` 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 `lstcsh`, 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
```

For more details on the `@` character, see "@ character" on page 556.

Limitations

A shell is a very complicated application by itself. `lstcsh` 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.

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 557.

tcsh version

`lstcsh` is based on `tcsh 6.03` (7 bit mode). It does not support the new features of the latest `tcsh`.

Starting lstcsh

Starting lstcsh

If you normally use some other shell, you can start `lstcsh` from the command-line.

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.

Exiting lstcsh

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.

Setting your login shell

Using csh

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`.

For example, `user3` can 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 Reference* for more details.

Examples

```
% hostname @hostD
<< remote execution on hostD >>
hostD
```

```
% hostname @/type==alpha
<< remote execution on hostB >>
hostB
```

@ character

`@`	`@` followed by nothing means that the command line is eligible for remote execution.
`@`*`host_name`*	`@` followed by a host name forces the command line to be executed on that host.
`@local`	`@` followed by the reserved word `local` forces the command line to be executed on the local host only.
`@/`*`res_req`*	`@` followed by `/` 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
```

Bringing a remote background task to the foreground

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

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_verbo
se, 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 550 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 `csh` 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 `tcsh` script with load-sharing disabled.

Running 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
  ```

  OR:

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