Platform LSF
Version 9 Release 1.1

Running Jobs

SC27-5307-01

Platform LSF
Version 9 Release 1.1

Running Jobs

SC27-5307-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 43.

First edition

This edition applies to version 9, release 1 of IBM Platform LSF (product number 5725G82) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About IBM Platform LSF. 1
Clusters, jobs, and queues 1
Hosts 3
LSF daemons 5
Batch jobs and tasks 7
Host types and host models 9
Users and administrators 9
Resources 10
Job lifecycle 12

Working with Jobs 15
Submitting jobs (bsub) 15
Modify pending jobs (bmod) 18
Modify running jobs 19
About controlling jobs 21
Using LSF with non-shared file space 27
operator 27
About resource reservation 28

Running MPI Jobs 31
Open MPI 31
Platform MPI 31
MVAPICH 31
Intel MPI and mpich2 32

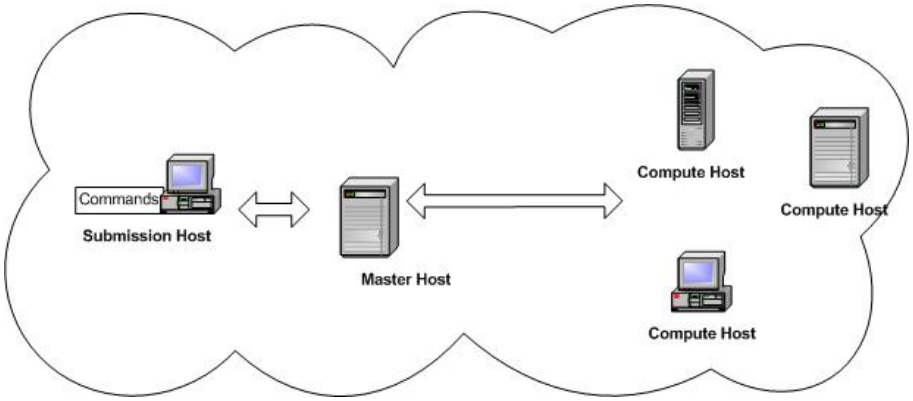
Monitoring Jobs. 37
View information about jobs 37
About displaying resource allocation limits 41

Notices 43
Trademarks 45

About IBM Platform LSF

"Clusters, jobs, and queues"
"Hosts" on page 3
"LSF daemons" on page 5
"Batch jobs and tasks" on page 7
"Host types and host models" on page 9
"Users and administrators" on page 9
"Resources" on page 10
"Job lifecycle" on page 12

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 a network of computing resources.

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

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

Commands

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

Configuration

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

Tip:

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

Job

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

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

Commands

- **bjobs** — View jobs in the system
- **bsub** — Submit jobs

Job slot

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

Commands

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

Configuration

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

Job states

LSF jobs have the following states:

- PEND — Waiting in a queue for scheduling and dispatch
- RUN — Dispatched to a host and running
- DONE — Finished normally with zero exit value
- EXIT — 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`

  Tip:

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

First-come, first-served scheduling (FCFS)

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

Hosts

A host is an individual computer in the cluster.

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

Commands

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

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

Submission host

The host where jobs are submitted to the cluster.

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

Client hosts and server hosts can act as submission hosts.

Commands:

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

Execution host

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

Commands

- **bjobs** — View where a job runs

Server host

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

Commands

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

Configuration

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

Client host

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

Commands

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

Configuration

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

Master host

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

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

Commands

- **lsid** — View the master host name

Configuration

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

LSF daemons

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

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, and resource availability. Sends scheduling decisions to **mbatchd**.

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

Commands

- **badmin hstartup** — Starts **sbatchd**
- **badmin hshutdown** — Shuts down **sbatchd**
- **badmin hrestart** — Restarts **sbatchd**

Configuration

- Port number defined in `lsf.conf`

res

- Remote Execution Server (**res**) running on each server host.

- Accepts remote execution requests to provide transparent and secure remote execution of jobs and tasks.

Commands

- **lsadmin resstartup** — Starts **res**
- **lsadmin resshutdown** — Shuts down **res**
- **lsadmin resrestart** — Restarts **res**

Configuration

- Port number defined in `lsf.conf`.

lim

- Load Information Manager (LIM) running on each server host.
- Collects host load and configuration information and forwards it to the master LIM running on the master host.
- Reports the information displayed by **lsload** and **lshosts**.
- Static indices are reported when the LIM starts up or when the number of CPUs (ncpus) change. Static indices are:
  - Number of CPUs (ncpus)
  - Number of disks (ndisks)
  - Total available memory (maxmem)
  - Total available swap (maxswp)
  - Total available temp (maxtmp)
  - Dynamic indices for host load collected at regular intervals are:
  - Hosts status (status)
  - 15 second, 1 minute, and 15 minute run queue lengths (r15s, r1m, and r15m)
  - CPU utilization (ut)
  - Paging rate (pg)
  - Number of login sessions (ls)
  - Interactive idle time (it)
  - Available swap space (swp)
  - Available memory (mem)
  - Available temp space (tmp)
  - Disk IO rate (io)

Commands

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

Configuration

- Port number defined in `lsf.conf`.

Master LIM

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

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

Commands

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

Configuration

- Port number defined in `lsf.conf`.

ELIM

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

pim

- Process Information Manager (PIM) running on each server host.
- Started by LIM, which periodically checks on **pim** and restarts it if it dies.
- Collects information about job processes running on the host such as CPU and memory used by the job, and reports the information to **sbatchd**.

Commands

- **bjobs** — View job information

Batch jobs and tasks

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

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 system-wide resource requirements for tasks
- `lsf.task.cluster` — Configure cluster-wide 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

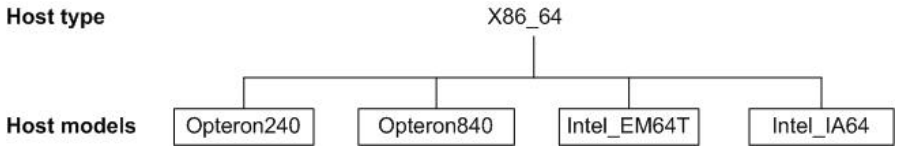
Configuration

- `lsf.task` — Configure system-wide resource requirements for tasks
- `lsf.task.cluster` — Configure cluster-wide 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 a host type of X86_64. Host models are Opteron240, Opteron840, Intel_EM64T, Intel_IA64.



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 startup.

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

Load thresholds

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

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

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

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

Commands

- **bhosts -l** — View suspending conditions for hosts
- **bqueues -l** — View suspending conditions for queues

- **bjobs -l** — View suspending conditions for a particular job and the scheduling thresholds that control when a job is resumed

Configuration

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

Runtime resource usage limits

Limit the use of resources while a job is running. Jobs that consume more than the specified amount of a resource are signalled or have their priority lowered.

Configuration

- lsb.queues — Configure resource usage limits for queues

Hard and soft limits

Resource limits specified at the queue level are hard limits while those specified with job submission are soft limits. See setrlimit(2) man page for concepts of hard and soft limits.

Resource allocation limits

Restrict the amount of a given resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to. If all of the resource has been consumed, no more jobs can be started until some of the resource is released.

Configuration

- lsb.resources — Configure queue-level resource allocation limits for hosts, users, queues, and projects

Resource requirements (bsub -R)

Restrict which hosts the job can run on. Hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it collects the load index values of all the candidate hosts and compares them to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

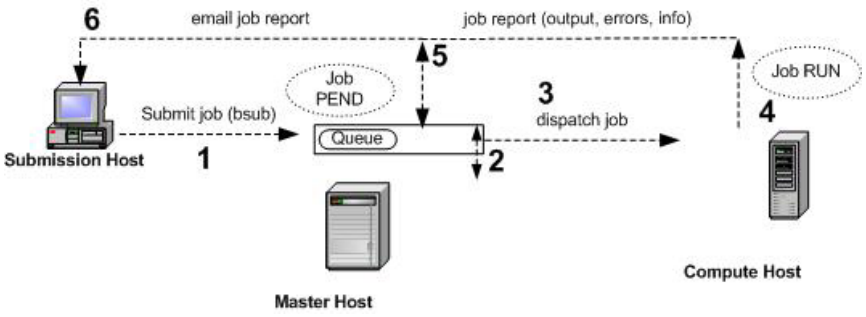
Commands

- **bsub -R** — Specify resource requirement string for a job

Configuration

- lsb.queues — Configure resource requirements for queues

Job lifecycle



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 the following:
   - 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 does the following:

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 and Linux, 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 the following information:

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

Working with Jobs

"Submitting jobs (bsub)"
"Modify pending jobs (bmod)" on page 18
"Modify running jobs" on page 19
"About controlling jobs" on page 21
"Using LSF with non-shared file space" on page 27
"operator" on page 27
"About resource reservation" on page 28

Submitting jobs (bsub)

"Submit a job"

Submit a job

To have your work run on the cluster, you must submit jobs. You have many options when submitting a job.

The following is the most basic way of submitting a job.

To submit a job, run **bsub**.
If you do not specify any options, the job is submitted to the default queue configured by the LSF administrator (usually queue normal)
For example, if you submit the job `my_job` without specifying a queue, the job goes to the default queue.

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

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

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

When you submit jobs, you can also specify an execution host or a range of hosts, a queue, and start and termination times, as well as a wide range of other job options.

"About submitting a job to a specific queue" on page 16
"View available queues" on page 16
"Submit a job to a queue" on page 16
"Submit a job associated with a project (bsub -P)" on page 17
"Submit a job associated with a user group (bsub -G)" on page 17
"Submit a job with a job name (bsub -J)" on page 17
"Submit a job to a service class (bsub -sla)" on page 17
"Submit a job under a job group (bsub -g)" on page 18

About submitting a job to a specific queue

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

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

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

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

View available queues

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

- To see available queues, run **bqueues**.
- To display only the queues that accept jobs from specific users or user groups, run **bqueues -u user_name**.
- To display only the queues that use specific host name or host group name to run jobs, run **bqueues -m host_name**.

Submit a job to a queue

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

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

  ```
  bsub -q "night" my_job
  ```

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

  ```
  bsub -q "priority" my_job
  ```

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

  ```
  bsub -q "idle" my_job
  ```

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

  ```
  bsub -q "short" my_job
  ```

  Tip:

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

- If your job requires a specific execution environment, you may need to submit it to a queue that has a particular job starter defined. LSF administrators are able

to specify a queue-level job starter as part of the queue definition; ask them for the name of the queue and configuration details.

Submit a job associated with a project (bsub -P)

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

To associate a project name with a job, run **bsub -P** *project_name*.
Project names can be up to 59 characters long.
On systems running IRIX 6, before the submitted job begins execution, a new array session is created and the project ID corresponding to the project name is assigned to the session.

Submit a job associated with a user group (bsub -G)

Note:

This task is only useful with fairshare scheduling.

You can specify any user group to which you belong. You must be a direct member of the specified user group.

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

To submit a job and associate it with a specified user group, run **bsub -G** *user_group*.
For example, to submit the job `myjob` associated to user group `special`:

```
bsub -G special myjob
```

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

You can associate a name with a job or job array.

Job names can contain up to 4094 characters.

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

Use **bsub -J** *job_name* to submit a job and assign a job name to it.
For example, to submit a job and assign the name `my_job`:

```
bsub -J my_job sleep 1000
```

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

You submit jobs to a service class as you would to a queue, except that a service class is a higher level scheduling policy that makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses.

The service class name where the job is to run is configured in `lsb.serviceclasses`. If the SLA does not exist or the user is not a member of the service class, the job is rejected.

Outside of the configured time windows, the SLA is not active, and LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with **-sla**.

Tip:

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.

To submit a job to a service class for SLA-driven scheduling, run **bsub -sla** *service_class_name*.
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 Duncan.

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

The job group does not have to exist before submitting the job.

To submit a job into a job group, run **bsub -g**. 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.

Modify pending jobs (bmod)

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

- To replace the job command line, run **bmod -Z "new_command"**.

  For example: **bmod -Z "myjob file" 101**.
- To change a specific job parameter, run **bmod -b**.

  The specified options replace the submitted options.

  The following example changes the start time of job 101 to 2:00 a.m.: bmod -b 2:00 101
- To reset an option to its default submitted value (undo a **bmod**), append the n character to the option name and do not include an option value.

  The following example resets the start time for job 101 back to its default value: **bmod -bn 101**

Modify the service class of a job

You can attach or detach a service class of a job after the job has been submitted.

Restriction:

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
- 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.

  bmod -sla Vernon 2307

  Attaches job 2307 to the service class Vernon.
- Use **bmod -slan** to detach a job from a service class.

  For example:

  bmod -slan 2307

  Detaches job 2307 from the service class Vernon.

Modify a job submitted to a job group

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.

Restriction:

The command **bmod -g** cannot be combined with other **bmod** options. It can only operate on pending jobs.

It cannot operate on running or finished jobs.

You 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.

To specify a job group path to move a job or a job array from one job group to another, run **bmod -g**.
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.

Modify the swap limit of a pending job

If a queue swap limit is defined, the value here cannot be greater than the value set for the queue.

- Modify the swap limit.

  bmod -v *swap_limit job_id*
- Remove the swap limit.

  bmod -vn *job_id*

If the job uses more than the new swap limit, the job is killed.

Modify running jobs

Once you submit a job and it is running, you can modify some of the job options, including resource reservation, CPU limit, memory limit, swap limit, and others.

Restriction:

You cannot modify remote running jobs in a MultiCluster environment.

- Modify resource reservation

- Modify other selected job options

Modify resource reservation

A job is usually submitted with a resource reservation for the maximum amount required. Use this command to decrease the reservation, allowing other jobs access to the resource.

Note:

You can modify additional job options by setting LSB_MOD_ALL_JOBS in `lsf.conf`.

Run **bmod -R** to modify the resource reservation for a running job.
For example, to set the resource reservation for job 101 to 25MB of memory and 50 MB of swap space:

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

Modify job options

Set **LSB_MOD_ALL_JOBS** is specified in `lsf.conf`. You must be the job owner or an LSF administrator to modify a running job.

Additionally, you must set the following parameters for each type of modification type:

- To modify the CPU limit of running jobs, **LSB_JOB_CPULIMIT=Y** must be defined in `lsf.conf`.
- To modify the memory limit of running jobs, **LSB_JOB_MEMLIMIT=Y** must be defined in `lsf.conf`.
- To modify the name of job error file for a running job, you must use **bsub -e** or **bmod -e** to specify an error file before the job starts running.

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

Restriction:

You cannot use these options in combination with other **bmod** options. An error message is issued and the modification fails if these options are used on running jobs in combination with other **bmod** options.

Run **bmod** with the appropriate option:

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

About controlling jobs

LSF controls jobs dispatched to a host to enforce scheduling policies or in response to user requests.

The LSF system performs the following actions on a job:

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

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

"Kill a job (bkill)"

"About suspending and resuming jobs (bstop and bresume)" on page 22

"Suspend a job" on page 22

"Resume a job" on page 23

"Move a job to the bottom of a queue (bbot)" on page 23

"Move a job to the top of a queue (btop)" on page 23

"Control jobs in job groups" on page 24

"Submit a job to specific hosts" on page 25

"Submit a job with specific resources" on page 25

"Queues and host preference" on page 25

"Specify different levels of host preference" on page 26

"Submit a job with resource requirements" on page 26

"Submit a job with SSH X11 forwarding" on page 26

Kill a job (bkill)

You can cancel a job from running or pending by killing it.

If your job is part of a job group, the command **bkill** only kills jobs in the job group you specify. It does not kill jobs in lower level job groups in the path.

To kill job 3421:

```
bkill 3421
```

1. 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
   ```

2. Use job ID 0 (zero) to terminate all jobs in a job group:

   ```
   bkill -g /risk_group 0
   ```

   "Forcing removal of a job from LSF"

Forcing removal of a job from LSF

If a job cannot be killed in the operating system, you can force the removal of the job from LSF.

The **bkill -r** command removes a job from the system without waiting for the job to terminate in the operating system. This sends the same series of signals as **bkill** without **-r**, except that the job is removed from the system immediately, the job is marked as EXIT, and job resources that LSF monitors are released as soon as LSF receives the first signal.

Use **bkill -r** to force the removal of the job from LSF.

About suspending and resuming jobs (bstop and bresume)

You can resume or suspend a job using the **bstop** and **bresume** commands.

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

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

If a user suspends a high priority job from a non-preemptive queue, the load may become low enough for LSF to start a lower priority job in its place. The load created by the low priority job can prevent the high priority job from resuming. This can be avoided by configuring preemptive queues.

The command **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`.

Allow users to resume jobs

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

Suspend a job

You can suspend or stop a job that is already running.

You must be an administrator or the user who submitted the job.

When you stop a job, it is suspended until you choose to resume it.

Suspending a job causes your job to go into USUSP state if the job is already started, or to go into PSUSP state if your job is pending.

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

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
```

Resume a job

You can resume a job that was suspended using **bstop**.

You must be the same user who suspended the job. If your job was suspended by the administrator, you cannot resume it; the administrator or `root` must resume the job for you.

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.

From the command line, run **bresume** and specify a job group path to resume suspended jobs in a job group.
For example, to resume job 3421, enter **bresume 3421**.

Move a job to the bottom of a queue (bbot)

Use **bbot** to move jobs relative to your last job in the queue.

You must be an LSF administrator or the user who submitted the job.

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 **bbot** command 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.

To move a job to the bottom of the queue, run **bbot**.
For example, **bbot 5311**.

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.

Move a job to the top of a queue (btop)

Use **btop** to move jobs relative to your first job in the queue.

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** command 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.

To move a job to the top of the queue, run **btop**.
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.

Tip: 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
```

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.

Control jobs in job groups

Stop jobs (bstop)

1. To suspend jobs in a job group, run **bstop -g** and specify a job group path.

   For example:

   ```
   bstop -g /risk_group 106
   Job <106> is being stopped
   ```

2. To suspend all jobs in a job group, use job ID 0 (zero).

   ```
   bstop -g /risk_group/consolidate 0
   Job <107> is being stopped
   Job <108> is being stopped
   Job <109> is being stopped
   ```

Resume jobs (bresume)

1. To resume suspended jobs in a job group, run **bresume -g** and specify a job group path.

   For example:

   ```
   bresume -g /risk_group 106
   Job <106> is being resumed
   ```

2. To resume all jobs in a job group, use job ID 0 (zero).

   ```
   bresume -g /risk_group 0
   Job <109> is being resumed
   Job <110> is being resumed
   Job <112> is being resumed
   ```

Terminate jobs (bkill)

The command **bkill** only kills jobs in the job group you specify. It does not kill jobs in lower level job groups in the path.

1. To terminate jobs in a job group, run **bkill -g** and specify a job group path.

For example:

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

2. To terminate all jobs in a job group, run job ID 0 (zero).

```
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
```

Delete job groups (bgdel)

The job group cannot contain any jobs.

To remove a job group, run **bgdel**.
For example:

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

The job group /risk_group and all its subgroups are deleted.

Submit a job to specific hosts

Specify which hosts a job can run on.

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

Note:

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

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

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

Submit a job with specific resources

Configure a new resource that your job runs with.

You must be the LSF administrator.

If you have applications that need specific resources, you should create a new Boolean resource and configure that resource for the appropriate hosts in the cluster.

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

Queues and host preference

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

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

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

Specify different levels of host preference

Submit a job with varying levels of host preferences.

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

Submit a job indicating a host preference.
For example,

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

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

Submit a job with resource requirements

Submit a job that requires specific resources to run.

Submit a job indicating a resource requirement.
For example, to submit a job which will run on Solaris 7 or Solaris 8:

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

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

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

Submit a job with SSH X11 forwarding

This task applies to UNIX execution and submissions hosts only. You cannot modify the **DISPLAY** environment variable.

You can submit a job to run with SSH X11 forwarding. These jobs run securely between the X-Server host, the submission host, and the execution host. Allows you to run jobs through an SSH client without having the SSH daemon installed on the X-Server host.

1. Log into the X-Server host.
2. Enable SSH X11 forwarding in your SSH client.
3. Use the SSH client to log into a submission host.
4. Run `bsub -XF` or `-XF -I`.

   If required, provide the password for public or private keys.

   The job is scheduled by LSF and runs. The **bsub** command waits until the job is dispatched.

   If interactive (**-I**), the job displays throughout the lifecycle of the job.

Using LSF with non-shared file space

LSF is usually used in networks with shared file space. When shared file space is not available, use the **bsub -f** command to have LSF copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

LSF attempts to run the job in the directory where the **bsub** command was invoked. If the execution directory is under the user's home directory, **sbatchd** looks for the path relative to the user's home directory. This handles some common configurations, such as cross-mounting user home directories with the **/net** automount option.

If the directory is not available on the execution host, the job is run in `/tmp`. Any files created by the batch job, including the standard output and error files created by the **-o** and **-e** options to **bsub**, are left on the execution host.

LSF provides support for moving user data from the submission host to the execution host before executing a batch job, and from the execution host back to the submitting host after the job completes. The file operations are specified with the **-f** option to **bsub**.

LSF uses the **lsrcp** command to transfer files. **lsrcp** contacts RES on the remote host to perform file transfer. If RES is not available, the UNIX **rcp** command is used. If **LSF_REMOTE_COPY_COMMAND** is specified in `lsf.conf`, **lsrcp** uses that command and any options to transfer files instead.

bsub -f

The **-f "[***local_file operator* **[***remote_file***]]"** option to the **bsub** command copies a file between the submission host and the execution host. To specify multiple files, repeat the **-f** option.

local_file

File name on the submission host.

remote_file

File name on the execution host.

The files *local_file* and *remote_file* can be absolute or relative file path names. You must specify at least one file name. When the file *remote_file* is not specified, it is assumed to be the same as *local_file*. Including *local_file* without the operator results in a syntax error.

operator

Operation to perform on the file. The operator must be surrounded by white space.

Valid values for *operator* are:

- >

  local_file on the submission host is copied to *remote_file* on the execution host before job execution. *remote_file* is overwritten if it exists.

- <

  remote_file on the execution host is copied to *local_file* on the submission host after the job completes. *local_file* is overwritten if it exists.

- <<

  remote_file is appended to *local_file* after the job completes. *local_file* is created if it does not exist.
- ><, <>

  Equivalent to performing the > and then the < operation. The file *local_file* is copied to *remote_file* before the job executes, and *remote_file* is copied back, overwriting *local_file*, after the job completes. <> is the same as ><

If the submission and execution hosts have different directory structures, you must ensure that the directory where *remote_file* and *local_file* will be placed exists. LSF tries to change the directory to the same path name as the directory where the **bsub** command was run. If this directory does not exist, the job is run in your home directory on the execution host.

You should specify *remote_file* as a file name with no path when running in non-shared file systems; this places the file in the job's current working directory on the execution host. This way the job will work correctly even if the directory where the **bsub** command is run does not exist on the execution host. Be careful not to overwrite an existing file in your home directory.

About resource reservation

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs do not consume the resources they require when they first start. Instead, they will typically use the resources over a period of time.

For example, a job requiring 100 MB of swap is dispatched to a host having 150 MB of available swap. The job starts off initially allocating 5 MB and gradually increases the amount consumed to 100 MB over a period of 30 minutes. During this period, another job requiring more than 50 MB of swap should not be started on the same host to avoid over-committing the resource.

You can reserve resources to prevent overcommitment by LSF. Resource reservation requirements can be specified as part of the resource requirements when submitting a job, or can be configured into the queue level resource requirements.

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
```

"View resource information" on page 29

"Submit a job with resource requirements" on page 29

"Submit a job with start or termination times" on page 29

"Submit a job with resource requirements" on page 26

"Submit a job with compute unit resource requirements" on page 29

View resource information

You can view host - or queue-level resource information.

1. View host-level resource information.
   a. To view the amount of resources reserved on each host, run **bhosts -l**.
   b. To view information about shared resources, run **bhosts -s**.
2. View the queue-level resource information.
   a. To see the resource usage configured at the queue level, run **bqueues -l**.

Submit a job with resource requirements

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

```
bsub -R "rusage[tmp=30:duration=30:decay=1]" myjob
```

LSF reserves 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.

Submit a job with start or termination times

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

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

If you do not want to start your job immediately when you submit it, specify a start time for LSF to start it.

You can also specify a time after which the job should be terminated.

1. Submit a job with a start time.

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

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

   ```
   bsub -t 11:12:20:30 myjob
   ```

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

Submit a job with resource requirements

Submit a job that requires specific resources to run.

Submit a job indicating a resource requirement.
For example, to submit a job which will run on Solaris 7 or Solaris 8:

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

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

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

Submit a job with compute unit resource requirements

Submit a job with compute unit resource requirements.

1. Submit a job to run on a certain number of compute units or fewer using the keyword maxcus in the resource requirement cu string.

   To submit a job which will run over 4 or fewer compute units of type rack for example:

   ```
   bsub -R "cu[type=rack:maxcus=4]" my_job
   ```

2. Submit a job to run balanced over the fewest possible compute units using the keyword balance in the resource requirement cu string.

   To submit a job which will run evenly distributed over the fewest compute units and use a total of 64 slots, for example:

   ```
   bsub -n 64 -R "cu[balance]" my_job
   ```

   Possible resource allocations (in order of preference) are:

   - 64 slots on 1 enclosure
   - 32 slots on 2 enclosures
   - 22 slots on 1 enclosure and 21 slots on 2 enclosures
   - 16 slots on 4 enclosures

3. Submit a job to run on compute units few or many available slots use the keyword pref.

   To submit a job to run on compute units with the fewest slots for example:

   ```
   bsub -R "cu[pref=minavail]" my_job
   ```

Running MPI Jobs

"Open MPI"
"Platform MPI"
"MVAPICH"
"Intel MPI and mpich2" on page 32

Open MPI

LSF must be installed and running. You must build Open MPI according to the Open MPI documentation for implement Open MPI with LSF.

Open MPI 1.3.2 and up is tightly integrated with LSF's **blaunch** functionality.

Run Open MPI jobs in LSF.

```
bsub -n2 -o %J.out -e %J.err mpiexec mympi.out
```

Platform MPI

Platform MPI (formerly HP-MPI) is integrated with LSF.

1. Set the **MPI_REMSH** environment variable.

   ```
   MPI_REMSH=blaunch;export MPI_REMSH
   ```

2. Run your job. For example:

   ```
   bsub -n 16 -R "span[ptile=4]" $MPI_ROOT/bin/mpirun -lsb_mcpu_hosts a.out
   ```

   To use Platform MPI with InfiniBand:

   ```
   bsub -n 16 -R "span[ptile=4]" $MPI_ROOT/bin/mpirun -lsb_mcpu_hosts -IBV
   a.out
   ```

MVAPICH

MVAPICH can be integrated with LSF.

1. Choose from two options:
   a. Change the MVAPICH source code (if you only want to run MVAPICH with LSF).

      Modify the MVAPICH source code: `RSH_CMD = 'blaunch'` and build the package.

   b. Write a wrapper script.

      Wrap `/usr/bin/rsh` on the first execution host or all candidate execution hosts for **blaunch** as follows:

      Example wrapper script:

      ```
      cat /usr/bin/rsh
      #!/bin/sh
      #
      # wrapper /usr/bin/rsh
      # blaunch is used when applicable
      #
      if [ -z "$LSF_BINDIR" \
           -o -z "$LSB_JOBID" \
           -o -z "$LSB_JOBINDEX" \
           -o -z "$LSB_JOBRES_CALLBACK" \
           -o -z "$LSB_DJOB_HOSTFILE" ]; then
          RSH="/usr/bin/rsh.bin"
      ```

```
else
    RSH=$LSF_BINDIR/blaunch
fi
$RSH $*
```

c. If you wrote a wrapper script, specify host file with a script.

Example:

```
cat run.mvapich
#! /bin/sh
#BSUB -n 2
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -R 'span[ptile=1]'
NUMPROC=`wc -l $LSB_DJOB_HOSTFILE|awk '{print $1}'`
mpirun_rsh -rsh -np $NUMPROC -hostfile $LSB_DJOB_HOSTFILE a.out
mpirun_rsh -rsh -np $LSB_DJOB_NUMPROC -hostfile $LSB_DJOB_HOSTFILE mympi
```

2. Run bsub.

For example, `bsub < run.mvapich`.

Intel MPI and mpich2

Intel MPI is a variation of MPICH2.This solution applies to either integration.

1. Create a wrapper script around **mpdboot**, without the daemonize option.

It should:

- loop all hosts and **blaunch mpd** without **-d** option in background
- at the end, check whether the **mpd** ring is constructed correctly
- exit 0 if correctly constructed, otherwise print out error

Example:

```
#!/usr/bin/env python2.3
"""
mpdboot for LSF
    [-f | --hostfile hostfile]
    [-i | --ifhn=alternate_interface_hostname_of_ip_address
     -f | --hostfile hostfile]
    [-h]
"""
import re
import string
import time
import sys
import getopt
from time import ctime
from os       import environ, path
from sys      import argv, exit, stdout
from popen2   import Popen4
from socket   import gethostname, gethostbyname
def mpdboot():
    # change me
    MPI_ROOTDIR="/opt/mpich2"
    #
    mpdCmd="%s/bin/mpd" % MPI_ROOTDIR
    mpdtraceCmd="%s/bin/mpdtrace" % MPI_ROOTDIR
    mpdtraceCmd2="%s/bin/mpdtrace -l" % MPI_ROOTDIR
    nHosts = 1
    host=""
    ip=""
    localHost=""
    localIp=""
    found = False
    MAX_WAIT = 5
    t1 = 0
```

```
    hostList=""
    hostTab = {}
    cols = []
    hostArr = []
    hostfile = environ.get('LSB_DJOB_HOSTFILE')
    binDir = environ.get('LSF_BINDIR')
    if environ.get('LSB_MCPU_HOSTS') == None \
        or hostfile == None \
        or binDir == None:
        print "not running in LSF"
        exit (-1)
    rshCmd = binDir + "/blaunch"
    p = re.compile("\w+_\d+\s+\(\d+\.\d+\.\d+\.\d+")
#
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hf:i:", ["help", "hostfile=", "ifhn="])
    except getopt.GetoptError, err:
        print str(err)
        usage()
        sys.exit(-1)
    fileName = None
    ifhn = None
    for o, a in opts:
        if o == "-v":
            version();
            sys.exit()
        elif o in ("-h", "--help"):
            usage()
            sys.exit()
        elif o in ("-f", "--hostfile"):
            fileName = a
        elif o in ("-i", "--ifhn"):
            ifhn = a
        else:
            print "option %s unrecognized" % o
            usage()
            sys.exit(-1)
    if fileName == None:
        if ifhn != None:
            print "--ifhn requires a host file containing 'hostname ifhn=alternate_interface_hostname_of_ip_address'\n"
            sys.exit(-1)
        # use LSB_DJOB_HOSTFILE
        fileName = hostfile
    localHost = gethostname()
    localIp = gethostbyname(localHost)
    pifhn = re.compile("\w+\s+\ifhn=\d+\.\d+\.\d+\.\d+")
    try:
        # check the hostfile
        machinefile = open(fileName, "r")
        for line in machinefile:
            if not line  or  line[0] == '#':
                continue
            line = re.split('#', line)[0]
            line = line.strip()
            if not line:
                continue
            if not pifhn.match (line):
                # should not have --ifhn option
                if ifhn != None:
                    print "host file %s not valid for --ifhn" % (fileName)
                    print "host file should contain 'hostname ifhn=ip_address'"
                    sys.exit(-1)
                host = re.split(r'\s+',line)[0]
                if cmp (localHost, host) == 0 \
                   or cmp(localIp, gethostbyname(host))== 0:
                    continue
                hostTab[host] = None
```

```
        else:
            # multiple blaunch-es
            cols = re.split(r'\s+\ifhn=',line)
            host = cols[0]
            ip = cols[1]
            if cmp (localHost, host) == 0 \
               or cmp(localIp, gethostbyname(host))== 0:
                continue
            hostTab[host] = ip
        nHosts += 1
        #print "line: %s" % (line)
    machinefile.close()
except IOError, err:
    print str(err)
    exit (-1)
# launch an mpd on localhost
if ifhn != None:
    cmd = mpdCmd + " --ifhn=%s " % (ifhn)
else:
    cmd = mpdCmd
print "Starting an mpd on localhost:", cmd
Popen4(cmd, 0)
# wait til 5 seconds at max
while t1 < MAX_WAIT:
    time.sleep (1)
    trace = Popen4(mpdtraceCmd2, 0)
    # hostname_portnumber (IP address)
    line = trace.fromchild.readline()
    if not p.match (line):
        t1 += 1
        continue
    strings = re.split('\s+', line)
    (basehost, baseport) = re.split('_', strings[0])
    #print "host:", basehost, "port:", baseport
    found = True
    host=""
    break
if not found:
    print "Cannot start mpd on localhost"
    sys.exit(-1)
else:
    print "Done starting an mpd on localhost"
# launch mpd on the rest of hosts
if nHosts < 2:
    sys.exit(0)
print "Constructing an mpd ring ..."
if ifhn != None:
    for host, ip in hostTab.items():
        #print "host : %s ifhn %s\n" % (host, ip)
        cmd="%s %s %s -h %s -p %s --ifhn=%s" % (rshCmd, host, mpdCmd, basehost, baseport, ip)
        #print "cmd:", cmd
        Popen4(cmd, 0)
else:
    for host, ip in hostTab.items():
        #print "host : %s ifhn %s\n" % (host, ip)
        hostArr.append(host +  " ")
    hostList = string.join(hostArr)
    #print "hostList: %s" % (hostList)
    cmd="%s -z \'%s\' %s -h %s -p %s" % (rshCmd, hostList, mpdCmd, basehost, baseport)
    #print "cmd:", cmd
    Popen4(cmd, 0)
# wait till all mpds are started
MAX_TIMEOUT = 300 + 0.1 * (nHosts)
t1 = 0
started = False
while t1 < MAX_TIMEOUT:
    time.sleep (1)
```

```
        trace = Popen4(mpdtraceCmd, 0)
        if len(trace.fromchild.readlines()) < nHosts:
            t1 += 1
            continue
        started = True
        break
    if not started:
        print "Failed to construct an mpd ring"
        exit (-1)
    print "Done constructing an mpd ring at ", ctime()
def usage():
    print __doc__
if __name__ == '__main__':
    mpdboot()
```

```
cat run.intelmpi
#! /bin/sh
#BSUB -n 2
#BSUB -o %J.out
#BSUB -e %J.err
NUMPROC=`wc -l $LSB_DJOB_HOSTFILE|awk '{print $1}'`
mpdboot.lsf
mpiexec -np $NUMPROC mympi.out
mpdallexit
```

2. Run **bsub**.

   For example, `bsub < run.intelmpi`.

Monitoring Jobs

"View information about jobs"
"About displaying resource allocation limits" on page 41

View information about jobs

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

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

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

"View unfinished jobs"
"View all jobs"
"View running jobs"
"View pending reasons for jobs" on page 38
"View suspending reasons for jobs" on page 38
"View detailed job information" on page 38
"View job group information" on page 38
"Monitor SLA progress" on page 39
"View job output" on page 39
"View chronological history of jobs" on page 39
"View history of jobs not listed in active event log" on page 40
"View job history" on page 40
"Update interval" on page 40
"Job-level information" on page 40

View unfinished jobs

Run **bjobs** to view the status of LSF jobs.
When no options are specified, **bjobs** displays information about jobs in the PEND, RUN, USUSP, PSUSP, and SSUSP states for the current user.

View all jobs

You can display information about jobs that are both running and those recently finished (PEND, RUN, USUSP, PSUSP, SSUSP, DONE, and EXIT statuses)

Run **bjobs -a** .
All your jobs that are still in the system and jobs that have recently finished are displayed.

View running jobs

You can display information about only jobs that are running (RUN status).

Run **bjobs -r**.
All your running jobs are displayed.

View pending reasons for jobs

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

1. Run **bjobs -p**.

   Displays information for pending jobs (PEND state) and their reasons. There can be more than one reason why the job is pending.

   The pending reasons alsodisplay the number of hosts for each condition.
2. To get specific host names along with pending reasons, **run bjobs -lp**.
3. To view the pending reasons for all users, run **bjobs -p -u all**.

View suspending reasons for jobs

When you submit a job, it may be held in the queue before it starts running and it may be suspended while running.

1. Run **bjobs -s**.

   Displays information for suspended jobs (SUSP state) and their reasons. There can be more than one reason why the job is suspended.

   The pending reasons alsodisplay the number of hosts for each condition.
2. To get specific host names along with suspend reasons, **run bjobs -ls**.
3. To view the suspend reasons for all users, run **bjobs -s -u all**.

View detailed job information

The **-l** option of **bjobs** displays detailed information about job status and parameters, such as the job's current working directory, parameters specified when the job was submitted, and the time when the job started running.

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

- Submission parameters
- Execution environment
- Resource usage

Run **bjobs -l**.

```
bjobs -l 7678
Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>, Command <verilog>
Mon Oct 28 13:08:11 2009: Submitted from host <hostD>,CWD <$HOME>, Requested Resources <type==any && swp>35>;
PENDING REASONS:
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;
SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut   pg    io   ls  it  tmp  swp  mem
loadSched  -      0.7  1.0    -    4.0   -    -   -   -    -    -
loadStop   -      1.5  2.5    -    8.0   -    -   -   -    -    -
```

View job group information

You can view information about jobs in job groups or view jobs by job group.

- To see all job groups, run **bjgroup**.
- To see jobs by job group, run **bjobs -g /***group_name*.

Monitor SLA progress

You can display the properties of service classes configured in `lsb.serviceclasses` and the dynamic state information for each service class.

1. Run `bsla`
2. Run `bacct -sla` to display historical performance of a service class.

View job output

You must be logged on as the job owner.

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

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

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

Run **bpeek** *job_id*.

For example:

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

View chronological history of jobs

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

- Use the **-t** option of **bhist** to display the events chronologically instead of grouping all events for each job.
- Use the **-T** option to select only those events within a given time range.

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

```
bhist -t -T 14:00,14:30
Wed Oct 22 14:01:25 2009: Job <1574> done successfully;
Wed Oct 22 14:03:09 2009: Job <1575> submitted from host to Queue , CWD , User , Project , Command ,
Requested Resources ;
Wed Oct 22 14:03:18 2009: Job <1575> dispatched to ;
Wed Oct 22 14:03:18 2009: Job <1575> starting (Pid 210);
Wed Oct 22 14:03:18 2009: Job <1575> running with execution home , Execution CWD , Execution Pid <210>;
Wed Oct 22 14:05:06 2009: Job <1577> submitted from host  to Queue, CWD , User , Project , Command ,
Requested Resources ;
Wed Oct 22 14:05:11 2009: Job <1577> dispatched to ;
Wed Oct 22 14:05:11 2009: Job <1577> starting (Pid 429);
Wed Oct 22 14:05:12 2009: Job <1577> running with execution home, Execution CWD , Execution Pid <429>;
Wed Oct 22 14:08:26 2009: Job <1578> submitted from host to Queue, CWD , User , Project , Command;
Wed Oct 22 14:10:55 2009: Job <1577> done successfully;
Wed Oct 22 14:16:55 2009: Job <1578> exited;
Wed Oct 22 14:17:04 2009: Job <1575> done successfully;
```

View history of jobs not listed in active event log

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

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

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

Run **bhist -n** *num_logfiles*.

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

View job history

You can check on the status of your job since it was submitted. The **bhist** command displays a summary of the pending, suspended, and running time of jobs for the user who invoked the command.

Run **bhist**.

1. Run **bhist -l** to display the time information and a complete history of scheduling events for each job.
2. Use **bhist -u all** to display a summary for all users in the cluster.

Update interval

The job-level resource usage information is updated at a maximum frequency of every SBD_SLEEP_TIME seconds.

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.

Job-level information

Job-level information includes:

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

About displaying resource allocation limits

The command **blimits** displays:

- Configured limit policy name
- Users (**-u** option)
- Queues (**-q** option)
- Hosts (**-m** option)
- Project names (**-P** option)

Resources that have no configured limits or no limit usage are indicated by a dash (-). Limits are displayed in a USED/LIMIT format. For example, if a limit of 10 slots is configured and 3 slots are in use, then **blimits** displays the limit for SLOTS as 3/10.

If limits MEM, SWP, or TMP are configured as percentages, both the limit and the amount used are displayed in MB. For example, **lshosts** displays maximum memory (maxmem) of 249 MB, and MEM is limited to 10% of available memory. If 10 MB out of are used, **blimits** displays the limit for MEM as 10/25 (10 MB USED from a 25 MB LIMIT).

Configured limits and resource usage for builtin resources (slots, mem, tmp, and swp load indices) are displayed as INTERNAL RESOURCE LIMITS separately from custom external resources, which are shown as EXTERNAL RESOURCE LIMITS.

Limits are displayed for both the vertical tabular format and the horizontal format for Limit sections. If a vertical format Limit section has no name, **blimits** displays NONAME*nnn* under the NAME column for these limits, where the unnamed limits are numbered in the order the vertical-format Limit sections appear in the `lsb.resources` file.

If a resource consumer is configured as all, the limit usage for that consumer is indicated by a dash (-).

PER_HOST slot limits are not displayed. The **bhosts** commands displays these as MXJ limits.

In MultiCluster, **blimits** returns the information about all limits in the local cluster.

"View information about resource allocation limits"

View information about resource allocation limits

Your job may be pending because some configured resource allocation limit has been reached. You can display the dynamic counters of resource allocation limits configured in the Limit sections in `lsb.resources`.

Run **blimits** to display the current resource usage, including any limits that may be blocking your job.

Notices

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Intellectual Property Law
Mail Station P300
2455 South Road,
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application

programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

LSF®, Platform, and Platform Computing are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA

SC27-5307-01