Inside Platform Lava

Version 1.0
September 2007
Comments to: doc@platform.com

Platform

We'd like to hear from you

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

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

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

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

Last update

July 31 2007

Contents

CHAPTER 1

Inside Your Cluster

Contents

Cluster Characteristics

The Lava master host daemons

The Lava master host is a Lava server host that acts as the overall coordinator for the cluster. All Lava daemons run on the master host. The lim on the master host is the master LIM. The master host is installed on the front-end node (`frontend-0`).

master LIM LIM on the master host. The master host is installed on the front-end node.

mbatchd Master Batch Daemon running on the master host (the front-end node). Started by the slave batch daemon, `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`.

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.

Default Lava directories

The following directories are owned by the primary Lava administrator and are readable by all cluster users.

Directory	Description	Example
LSF_CONFDIR	Lava configuration directory	`/opt/lava/conf/`
LSB_CONFDIR	Lava batch configuration directory	`/opt/lava/conf/lsbatch/`
LSB_SHAREDIR	Lava batch job history directory	`/opt/lava/work/`
LSF_LOGDIR	Server daemon error logs, one for each Lava daemon	`/opt/lava/log/`

Four important Lava configuration files

Lava configuration is administered through several configuration files, which you use to modify the behavior of your cluster. The four most important files you will work with are the following files, which are installed on the master host (front-end node):

- `LSF_CONFDIR/lsf.conf`
- `LSF_CONFDIR/lsf.cluster.lava`
- `LSF_CONFDIR/lsf.shared`
- `LSB_CONFDIR/lava/configdir/lsb.queues`

These files are created on the front-end node during the Lava installation.

All the files are owned by `root`. The files are readable by all cluster users. You can change ownership of all the files to the Lava administrator.

lsf.conf

The most important file in Lava. It contains the paths to the Lava configuration directories, log directories, libraries, and other global configuration information.

A version of `lsf.conf` file is also installed on each compute host. It shows the location of the log directory and `conf` directory on the master host (front-end node).

lsf.cluster.lava

Defines the host name, model, and type of the master host (on the front-end node). It also defines the user name of the Lava administrator.

lsf.shared

This file is like a dictionary that defines all the keywords used by the Lava cluster. You can add your own keywords to specify the names of resources or host types.

Note that LSF_SERVERDIR is not a shared directory.

lsb.queues

Defines the Lava batch queues and their parameters for one Lava cluster.

Cluster name

The name of the cluster is `lava`. This name is part of the name of the `/opt/lava/conf/lsf.cluster.lava` file:

Lava hosts

- The Lava master host is configured in the `Hosts` section of `LSF_CONFDIR/lsf.cluster.lava`.
- The master host on the front-end node is dynamically configured as a Lava server host. This is indicated by `1` in the server column of the Hosts section of `LSF_CONFDIR/lsf.cluster.lava`.
- Host types installed in your cluster are listed in the `Hosts` section of `LSF_CONFDIR/lsf.cluster.lava`. The master host is configured by default. You can also add your compute hosts to this section.

  Before you configure your resources, you must add your compute hosts to the Hosts section of LSF_CONFDIR/lsf.cluster.lava.

Restarting and Reconfiguring Lava Daemons

Restarting the whole cluster

Lava starts automatically in your Platform OCS cluster. If you need to restart your cluster, you must restart both the master host (on the front-end node) and all the compute hosts.

To change configuration on the master host, you do not need to restart the whole cluster. You can simply restart the Lava daemons on the master host. (See "Reconfiguring the cluster" on page 10.)

When you restart the cluster, you must restart Lava individually on pvfs-io and compute-pvfs hosts.

To restart the cluster:

1. Log on to the master host (on the front-end node) as `root`:
2. Restart Lava on the master host (front-end node):

   ```
   # /etc/init.d/lava stop
   # /etc/init.d/lava start
   ```

3. Restart Lava on the compute hosts:

   ```
   # cluster-fork /etc/init.d/lava stop
   # cluster-fork /etc/init.d/lava start
   ```

4. Restart Lava on individual hosts, such as pvfs-io and compute-pvfs hosts:

   ```
   # ssh hostname /etc/init.d/lava stop
   # ssh hostname /etc/init.d/lava start
   ```

   For example:

   ```
   # ssh compute-pvfs-0 /etc/init.d/lava stop
   # ssh compute-pvfs-0 /etc/init.d/lava start
   ```

Restarting the master host

If you need to restart the Lava daemons on the master host without restarting the whole cluster, run the following commands:

1. Run `badmin hshutdown` to shut down the slave batch daemon (`sbatchd`) on the master host. For example:

   ```
   # badmin hshutdown frontend-0
   ```

2. Restart `mbatchd`:

   ```
   # badmin reconfig
   ```

   This causes `mbatchd` and `mbschd` to exit. The `mbatchd` cannot be restarted, because `sbatchd` is shut down. All Lava 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 the master host

If you have edited the configuration files, and do not need to recognize new hosts or remove hosts, you can reload the configuration files without restarting the cluster:

```
# badmin reconfig
```

For more information on changing your configuration, see “Reconfiguring the cluster” on page 10.

Controlling daemons

To control all daemons in the cluster, you must:

- Be logged on as root.
- Be able to run `rsh` or `cluster-fork` commands across all Lava hosts without having to enter a password.

  `rsh` must be enabled.

  The shell command specified by LSF_RSH in `lsf.conf` is used before `rsh` is tried.

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

Daemon	Action	Command	Permissions
`sbatchd`	Start	badmin hstartup [host_name ... | all]	Must be root
	Restart	badmin hrestart [host_name ... | all]	Must be root or the Lava administrator for other commands.
	Shut down	badmin hshutdown [host_name ... | all]	
`mbatchd` `mbschd`	Restart	badmin reconfig	Must be root or the Lava administrator for these commands
	Shut down	1 badmin hshutdown 2 badmin reconfig	
	Reconfigure	badmin reconfig	
RES	Start	lsadmin resstartup [host_name ... | all]	Must be root
	Shut down	lsadmin resshutdown [host_name ... | all]	Must be the Lava administrator for other commands
	Restart	lsadmin resrestart [host_name ... | all]	
LIM	Start	lsadmin limstartup [host_name ... | all]	Must be root
	Shut down	lsadmin limshutdown [host_name ... | all]	Must be the Lava administrator for other commands
	Restart	lsadmin limrestart [host_name ... | all]	
	Restart all in cluster	lsadmin reconfig	
All Lava daemons	Restart the whole cluster	lava stop lava start	Must be root

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

Reconfiguring the cluster

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

- `lsadmin reconfig`
- `badmin reconfig`

The reconfiguration commands you use depend on which files you change in Lava. 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.params`	`badmin reconfig`	reloads configuration files
`lsb.queues`	`badmin reconfig`	reloads configuration files
`lsf.cluster.lava`	`lsadmin reconfig` AND `badmin reconfig`	reconfigures LIM, reloads configuration files, and restarts `mbatchd`
`lsf.conf`	`lsadmin reconfig` AND `badmin reconfig`	reconfigures LIM and reloads configuration files, and restarts `mbatchd`
`lsf.shared`	`lsadmin reconfig` AND `badmin reconfig`	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 Lava 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 <compute-0-0> ...... done
Restart LIM on <compute-0-1> ...... done
Restart LIM on <compute-0-2> ...... 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.

Cluster Administrators

Primary cluster administrator

Required. The first cluster administrator is specified during installation as `root`. You can change the primary administrator in the `lsf.cluster.lava` file. The primary Lava administrator account owns the configuration and log files. The primary administrator has permission to perform cluster-wide operations, change configuration files, reconfigure the cluster, and control jobs submitted by all users.

Cluster administrators

Optional.

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 Lava administrator except that they do not have permission to change configuration files.

Adding cluster administrators

1. In the `ClusterAdmins` section of `LSF_CONFDIR/lsf.cluster.lava`, specify the list of cluster administrators following ADMINISTRATORS, separated by spaces. The first administrator in the list is the primary Lava administrator. All others are cluster administrators. You can specify user names and group names. For example:

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

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

Mail Notification

When a batch job completes or exits, Lava 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
- Lava 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`.

Controlling the 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

Lava sets LSB_MAILSIZE to the approximate size in KB of the email containing job output information. LSB_MAILSIZE is not recognized by the Lava 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.

For more information on mail notification, see the Platform Lava man pages 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`.

Managing Users, Hosts, and Queues

Making your cluster available to users

To set up the Lava environment for your users, use the following two shell files:

- `LSF_CONFDIR/cshrc.lsf` (for `csh`, `tcsh`)
- `LSF_CONFDIR/profile.lsf` (for `sh`, `ksh`, or `bash`)

Make sure all Lava users include one of these files at the end of their own `.cshrc` or `.profile` file, or run one of these two files before using Lava.

For csh or tcsh

Add `cshrc.lsf` to the end of the `.cshrc` file for all users:

- Copy the `cshrc.lsf` file into `.cshrc`

OR

- Add a line similar to the following to the end of `.cshrc`:

```
source /opt/lava/conf/cshrc.lsf
```

For sh, ksh, or bash

Add `profile.lsf` to the end of the `.profile` file for all users:

- Copy the `profile.lsf` file into `.profile`

OR

- Add a line similar to following to the end of `.profile`:

```
. /opt/lava/conf/profile.lsf
```

Controlling hosts

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

Closing a host

```
# badmin hclose compute-0-0
Close <compute-o-o> ...... 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

```
# badmin hopen compute-0-0
Open <hostB> ...... done
```

Dispatch windows

A dispatch window specifies one or more time periods during which a host will receive new jobs.

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.

For information on dispatch windows for queues, see “Controlling when jobs run” on page 21.

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.

1 Log on as the Lava administrator to any host in the cluster.

2 Edit `lsf.shared`:

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

```
Begin HostType
TYPENAME                        # Keyword
DEFAULT
LINUX86
LINUX64
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.

```
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).
Opteron848 60.0 (x15_3604_AMDOpterontmProcessor848)
Intel_IA64 12.0 (ia64 IA64)
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

Lava 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 Lava services are defined automatically in the `lsf.conf` file during installation.

If you find any registration conflicts, change your service port numbers as follows:

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 reconfig` to restart `mbatchd`.
7. Restart all the daemons in the cluster.

Matching host names and addresses

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

Lava 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.

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.

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

Other names for the same host are called aliases.

Lava 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 Lava, but Lava always displays the official name.

Host name services

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

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

Hosts that have more than one network interface usually have one Internet address for each interface. Such hosts are called *multi-homed hosts*. Lava identifies hosts by name, so it needs to match each of these addresses with a single host name.

To match each address with a host name, the host name information must be configured so that all of the Internet addresses for a host resolve to the same name.

This can be done in one of the following ways:

- Modify the system hosts file (`/etc/hosts`) and the changes will affect the whole system
- Create a Lava hosts file (`LSF_CONFDIR/hosts`) and Lava 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. This configuration can confuse Lava, because there is no way to determine that the two different names (or addresses) refer to the same host. Lava 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 Lava hosts file

If your Lava 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 Lava to use.

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

In the Lava `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 configurations

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
```

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 Lava 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 a Lava hosts file and configure these addresses using a unique host name in that file.

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. Lava (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 Lava can resolve this case, you do not need to create a Lava 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 Lava 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.

Controlling queues

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

Adding a queue

1. Log on as the Lava administrator to the front-end host.
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

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 Lava administrator uses the `bswitch` command to switch the jobs into regular queues. Jobs in other queues are not affected.

The following examples use queues named `night` and `idle`.

1. Log on as `root` or the Lava administrator to 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. In the following example, 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  Nov 21 18:16
   5310  user5 PEND   night    hostA     hostC        job10  Nov 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 you want to remove.
5. Save the changes to `lsb.queues`.
6. Run `badmin reconfig` to reconfigure `mbatchd`.

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
```

Configuring automatic job requeue

You can configure automatic job requeue to kill and requeue a job while it is running or when it is suspended.

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.

To manually requeue a job, see the instructions in *Running Jobs with Platform Lava*.

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 pend forever.

Configuring automatic job rerun for a queue

Enable automatic job rerun if you want to requeue and rerun a job when the execution host goes down or when the Lava system fails while the job is running. Rerunnable jobs do not rerun if the job fails.

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 Lava administrator. (To submit a rerunnable job, see the instructions in *Running Jobs with Platform Lava*.)

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

Example

```
RERUNNABLE = yes
```

Controlling when jobs run

Dispatch and run windows are time windows that control when Lava 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, Lava 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, Lava suspends running jobs, but new jobs can still be submitted to the queue.

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 windows:

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 dispatch windows.

You can also configure dispatch windows for a host, by setting DISPATCH_WINDOW in lsb.hosts and specifying one or more time windows. If no host dispatch window is configured, the window is always open.

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:

```
badmin reconfig
```

4. Run `bqueues -l` to display the run windows.

Error and Event Logging

System directories and log files

Lava uses directories for temporary work files, log files, and transaction files and spooling.

Lava keeps track of all jobs in the system by maintaining a transaction log in the work subtree. The Lava log files are found in the directory `/opt/lava/work/lava/logdir`

This is not a shared directory. It is not shared with the compute hosts, only the master candidate hosts.

Current job states

Lava 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. Lava keeps track of everything associated with the job in the `lsb.events` file. 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.

Do not remove or modify the current `lsb.events` file. Removing or modifying the `lsb.events` file could cause batch jobs to be lost.

History

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 refers to the `lsb.events.`*`n`* files.

Job scripts

When a user issues a `bsub` command from a shell prompt, Lava collects all 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 Lava and should not be modified by anyone.

Log directory permissions and ownership

Ensure that the LSF_LOGDIR directory is writable by `root`. The Lava administrator must own LSF_LOGDIR.

Managing error logs

Error logs maintain important information about Lava operations. When you see any abnormal behavior in Lava, you should first check the appropriate error logs to find out the cause of the problem.

Lava log files grow over time. These files should occasionally be cleared, either by hand or with automatic scripts.

Daemon error log

Lava 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 Lava 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 Lava 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*

Lava 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 Lava 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`). Look for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`.

Monitoring Your Cluster

Viewing cluster information

Lava provides commands for users to get information about the cluster. Cluster information includes the cluster master host, cluster name, cluster resource definitions, and cluster administrator.

To view the ...	Run ...
Version of Lava	`lsid`
Cluster name	`lsid`
Current master host	`lsid`
Cluster administrators	`lsclusters`

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

```
# lsid
Platform Lava 1.0, August 30, 2007
Copyright 1992-2007 Platform Computing Corporation

My cluster name is lava
My master name is frontend-0.public
```

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.

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.

Viewing host information

Lava uses some or all of the hosts in a cluster as execution hosts. The host list is configured by the Lava 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`

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 status of the host about its ability to run batch jobs:

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.

bhosts -l Displays the closed reasons. A closed host will not accept new batch jobs:

Status	Description
`closed_Adm`	A Lava 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_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`	A Lava administrator or root explicitly locked the host using lsadmin limlock. Running jobs are suspended (SSUSP). Use lsadmin limunlock to unlock LIM on the local host.
`closed_Wind`	Host is closed by a dispatch window defined in lsb.hosts. Running jobs are not affected.

lsload Displays the current state of the host about its ability to run batch jobs and remote tasks:

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.lava, 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.lava, displayed by lshosts -l).
`lockU`	Will not accept new batch jobs or remote tasks. A Lava administrator or root explicitly locked the host (i.e., lsadmin limlock). Running jobs are not affected.
`unavail`	Host is down, or LIM is unavailable.

To view all hosts in the cluster

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

To view detailed 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.

To view host load by host

The `lsload` command reports the current status and load levels of hosts in a cluster. The `lshosts -l` command shows the load thresholds.

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

Run `lsload` to see load levels for each host.

To view host architecture

A Lava 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 Lava administrator in the Lava 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.

To view host history

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

To view host model and type

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

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

To view host dispatch windows

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

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 running jobs, and the number of suspended jobs.

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 states can be changed by a Lava administrator or `root`.

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

To view available queues

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.

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

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

To view 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.

To view the history of state changes in a queue

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

To view queue administrators

Use `bqueues -l` for the queue.

To view information about run windows

Use `bqueues -l` to display information about queue run windows.

To view queue dispatch windows

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

CHAPTER 2

Working with Resources and Resource Requirements

Contents

Resource Classifications

The Lava system uses built-in and configured resources to track job resource requirements and schedule jobs according to the resources available on individual hosts.

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 Lava, 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 that is configured to share it, but doing so affects its value as seen by other hosts.

Note: Before you can specify resources or add your own configured resources, you must define your hosts in the Host section of lsf.cluster.lava. By default, the compute hosts are added to the cluster dynamically and are not defined in lsf.cluster.lava. Follow the example definitions in lsf.cluster.lava.

Boolean resources

Boolean resources have a value of one (1) if they are defined for a host, and zero (0) if they are not defined for the host. Use Boolean resources to configure host attributes to be used in selecting hosts to run jobs. For example:

- Machines may have different types and versions of operating systems.
- Machines may play different roles in the system, such as file server or compute server.
- Some machines may have special-purpose devices needed by some applications.
- Certain software packages or licenses may be available only on some of the machines.

Specify a Boolean resource in a resource requirement selection string of a job, to select only hosts that can run the job.

Some examples of Boolean resources:

Resource Name	Describes	Meaning of Example Name
`cs`	Role in cluster	Compute server
`fs`	Role in cluster	File server
`linux64`	Operating system	Linux operating system
`frame`	Available software	FrameMaker license

Use a boolean resource to specify a host

Usually, to indicate that a job must run on one of a number of specified hosts, you use the `bsub -m "hostA hostB ..."` option. By specifying a single host, you can force your job to wait until that host is available and then run on that host.

If you have applications that need specific resources, it is more flexible to create a new Boolean resource and configure that resource for the appropriate hosts in the cluster. This must be done by the Lava administrator. If you specify a host list using the `-m` option of `bsub`, you must change the host list every time you add a new host that supports the desired resources. By using a Boolean resource, the Lava administrator can add, move, or remove resources without forcing users to learn about changes to resource configuration.

Load indices

Load indices are built-in resources that measure the availability of dynamic, non-shared resources on hosts in the Lava cluster.

Load indices built into the LIM are updated at fixed time intervals.

External load indices are defined and configured by the Lava 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 Lava 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 a Lava administrator or root.
`unavail`	The host is down or the LIM on the host is not running or is not responding.

CPU run queue lengths

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.

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. Lava 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

Lava 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

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

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.

Interactive idle time

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.

Temporary directories

The `tmp` index is the space available in MB on the file system that contains the temporary directory (`/tmp`).

Swap space

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

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. Lava 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. (For information on ELIMs, see “External Load Indices and ELIM” on page 39.)

I/O rate

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.

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 Lava 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 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

The CPU factor (`cpuf`) 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 Lava administrator. For multiprocessor hosts, the CPU factor is the speed of a single processor; Lava automatically scales the host CPU load to account for additional processors.

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 that 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, a shared resource from one machine affects the availability of that resource as seen by other machines. One value for the entire cluster measures the utilization of the shared resource, but each host-based resource is measured separately.

Lava does not contain any built-in shared resources. All shared resources must be configured by the Lava 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 Lava.

- A shared resource cannot be used as a load threshold in the `Hosts` section of the `lsf.cluster.lava` file.
- A shared resource cannot be used in the `loadSched/loadStop` thresholds, or in the `STOP_COND` parameter in the queue definition in the `lsb.queues` file.

  For information on `loadSched`, `loadStop`, and `STOP_COND`, see "Configuring Load Thresholds" on page 53.

Configuring Your Own Resources

Lava schedules jobs based on available resources. There are many resources built into Lava, but you can also add your own resources, and then use them the 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 InfiniBand, while others are only connected to Ethernet, then you probably want to define a resource called `infini` and associate the `infini` resource with machines connected to InfiniBand. This way, users can specify resource `infini` if they want their jobs to run on machines connected to InfiniBand.

Adding new resources to your cluster

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

1. Log on to any host in the cluster as the Lava 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 the lsf.shared resource section” on page 36.
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.lava`.
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 the lsf.cluster.lava resourcemap section” on page 37.
5. Reconfigure your cluster.

Configuring the 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 Lava 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 the lsf.cluster.lava resourcemap section

Resources are associated with the hosts for which they are defined in the `ResourceMap` section of `lsf.cluster.lava`.

For each resource, you must specify the name and the hosts that have it.

Make sure that the hosts that have the resources you want to configure are defined in the `Host` section of `lsf.cluster.lava`. By default, the compute hosts are added to the cluster dynamically and are not defined in `lsf.cluster.lava`.

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:

- Five units of the `verilog` resource are defined on `host3` only (all hosts except `host1` and `host2`).
- Two units of the `synopsys` resource are shared between `host1` and `host2`. Two more units of the `synopsys` resource are defined on `host3` (shared among all the remaining hosts in the cluster).
- One unit of the `console` resource is defined on each host in the cluster (assigned explicitly). One 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.
- The keyword `others` refers to all hosts not otherwise listed in the instance.
- The keyword `default` refers to each host in the cluster, individually.

External Load Indices and ELIM

The Lava 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 Lava 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 Lava uses ELIM for external resource collection

The values of static external resources are specified through the `lsf.cluster.lava` 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.lava` 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.lava` 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.lava` 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; otherwise, it is undefined. 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 convention:

`LSF_SERVERDIR/elim.`*application*

For example, `elim.license`

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.

Note that LSF_SERVERDIR is not a shared directory.

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.

Debugging an ELIM

Set the parameter `LSF_ELIM_DEBUG=y` in the Parameters section of `lsf.cluster.lava` 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.lava` to configure how long LIM waits before restarting the ELIM.

Use the parameter `LSF_ELIM_RESTARTS=`*`integer`* in the Parameters section of `lsf.cluster.lava` to limit the number of times an ELIM can be restarted.

See the Platform Lava man pages for more details on these parameters.

Configuring 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 Lava 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.

Default configuration

If a job has no resource requirements, Lava places it on a host of the same type as the submission host (`type==local`). However, if a job has string or Boolean resource requirements specified and the host type has not been specified, Lava places the job on any host (`type==any`) that satisfies the resource requirements.

When to configure resource requirements

To override the Lava defaults, specify resource requirements explicitly. Resource requirements can be set for queues, for individual applications, or for individual jobs.

A resource requirement is an expression that contains resource names and operators.

Defining resource requirements for a queue

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`.

Example

```
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.

To specify resource requirements for a specific job, see *Running Jobs with Platform Lava*.

Resource Requirement Strings

Most Lava commands accept a `-R res_req` argument to specify resource requirements.

A resource requirement string describes the resources a job needs. Lava uses resource requirements to select hosts for remote execution and job execution.

How queue and job resource 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.
- An `order` section defined at the queue level is ignored if different `order` requirements are specified at the job level. The default `order` string is `r15s:pg`.

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.

Syntax **`select[`***`selection_string`***`] order[`***`order_string`***`]`**

The square brackets must be typed as shown.

The section names are `select` and `order`.

If no section name is given, 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.

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 0 as logical FALSE. Boolean resources have a value of 1 if they are defined for a host, and 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 for `bsub` is `type==local` unless a string or Boolean resource is specified, in which case it is `type==any`.

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

Example

```
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 Lava command. For example, when submitting a Lava job that requires a certain amount of shared scratch space, you might submit the job as follows:

```
$ bsub -R "avail_scratch > 200 && swap > 50" myjob
```

The above assumes that all hosts in the cluster have access to the shared scratch space. The job 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 Lava cluster have access to the scratch space. To exclude hosts that cannot access a shared resource, the `defined(`*`resource_name`*`)` function must be specified in the resource requirement string.

For example:

```
$ bsub -R "defined(avail_scratch) && avail_scratch > 100 && swap > 100" myjob
```

would exclude any hosts that cannot access the scratch resource. The Lava 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 (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`

Monitoring 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
server          Boolean  N/A    Lava server host
cserver         Boolean  N/A    Compute server
fserver         Boolean  N/A    File server
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

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 five 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 that share this resource. The `lshosts -s` command displays static shared resources. The `lsload -s` command displays dynamic shared resources.

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.

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  mem
loadSched    -      -    -     -   -   -   -   -   -    -    -
loadStop     -      -    -     -   -   -   -   -   -    -    -
```

Viewing information about load indices

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
```

Viewing resource requirements for a queue

Use `bqueues -l` to view resource requirements (RES_REQ) defined for the queue.

Viewing resource requirements for a job

Use `bjobs -l` to view resource requirements defined for the job:

After a job is finished, use `bhist -l` to view resource requirements defined for the job:

CHAPTER 3

Configuring Job Controls

Contents

Configuring 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 for a queue by the Lava administrator (`lsb.queues`) or for a job at submission time.

Limits specified at the queue level are *hard* limits, while those specified with job submission are *soft* limits.

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
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

How resource usage limits are prioritized

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 Lava 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`.

Specifying resource usage limits

Queues can enforce resource usage limits on running jobs. Lava supports most of the limits that the underlying operating system supports. In addition, Lava also supports a few limits that the underlying operating system does not support.

Specify queue-level resource usage limits using parameters in `lsb.queues`.

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 Lava.

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

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
```

To specify resource usage limits for a job, see *Running Jobs with Platform Lava*.

Setting the CPU time and run time limits

To set the CPU time limit and run time limit for jobs in a platform-independent way, Lava 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.

Normalization host

If no host or host model is given with the CPU time or run time, Lava selects a host as follows (in order of preference):

- The default CPU time normalization host if defined at the queue level (DEFAULT_HOST_SPEC in `lsb.queues`)
- The default CPU time normalization host if defined at the cluster level (DEFAULT_HOST_SPEC in `lsb.params`)
- 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).

Configuring Load Thresholds

You can configure load thresholds 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.

- loadStop

  The condition that determines when running jobs should be suspended.

Thresholds can be configured for each queue, for each host, or a combination of both. 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.

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 normalized 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 suspending conditions

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.lava` 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
```

Configuring Job Control Actions

After a job is started, it can be killed, suspended, or resumed by the system, a Lava user, or Lava administrator. Lava job control actions cause the status of a job to change.

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 it is:
  - Suspended when a run window closes
  - Killed when the RUNLIMIT is reached

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`.

Configuring job controls

The JOB_CONTROLS parameter in `lsb.queues` has the following format:

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

When Lava 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.

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.
- If the TERMINATE action is CHKPNT, then the job is checkpointed and killed.

command A `/bin/sh` command line. Do not quote the command line inside an action definition.

See the Platform Lava man pages for information about the job control parameters in the `lsb.queues` file.

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 Lava waits for the job to end. If the job is not killed by the TERMINATE action, it remains suspended indefinitely.

TERMINATE_WHEN parameter

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 in `lsb.queues` to configure the queue to invoke the TERMINATE action instead of SUSPEND.

See the Platform Lava man pages for information about the TERMINATE_WHEN parameter in the `lsb.queues` file.

Syntax

```
TERMINATE_WHEN = WINDOW | LOAD
```

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
```

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 or not.

  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.

LSB_SIGSTOP parameter

Use LSB_SIGSTOP in `lsf.conf` 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 Lava man pages for information about LSB_SIGSTOP in 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]

Configuring Pre-Execution 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.

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 that has a pre-execution command, Lava 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, Lava 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. (See "Configuring automatic job requeue" on page 20.)

  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 329 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.

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'
```

```
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

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
```

Configuring Job Starters for Queues

Lava 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.

Queue-level job starters have no effect on interactive jobs, unless the interactive job is submitted to a queue as an interactive batch job.

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

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, Lava 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"
```

Index

N

O

P

Q

R