Installing and Configuring Platform LSF for SLURM

SLURM Version 1.2.25
Platform LSF Version 7 Update 5
January 30 2009

Comments to: doc@platform.com

Contents

- "Platform LSF for SLURM Overview" on page 2
- "Installing a New Platform LSF for SLURM Cluster" on page 8
- "Configuring Platform LSF for SLURM" on page 11
- "Operating Platform LSF for SLURM" on page 17
- "Submitting and Monitoring Jobs" on page 25
- "Command Reference" on page 35
- "File Reference" on page 37

Platform LSF for SLURM Overview

Contents

- "About Platform LSF for SLURM"
- "About this document"
- "Assumptions and limitations"

About Platform LSF for SLURM

Simple Linux Utility for Resource Management (SLURM)

SLURM is a resource management system suitable for use on large and small Linux clusters. It was jointly developed by Lawrence Livermore National Laboratory (LLNL), HP, Bull, and Linux NetworX. As a resource manager, SLURM allocates exclusive or non-exclusive access to resources on compute nodes) for users to perform work, and provides a framework to start, execute and monitor work (normally parallel jobs) on the set of allocated nodes.

A SLURM system consists of two daemons:

- `slurmctld`—runs on a node with the resource manager role as a central "controller" daemon that monitors node state and allocates nodes to jobs. Primary and backup `slurmctld` can run on separate resource manager nodes.
- `slurmd`—runs as root on nodes with compute role to export control to SLURM for starting and managing user jobs.

The SLURM configuration file (`slurm.conf`) must be available on each node of the system. Use the SLURM `scontrol reconfig` command to see the current SLURM configuration.

SLURM terminology

- **Node**—the basic hardware unit in a computing cluster. One node is one computer running Linux, configured with one or more role:
  - Compute role provides services to run user tasks. In this document, *node* means node with compute role. Compute role nodes are monitored and controlled by SLURM.
  - Resource manager role provides administrative and operating system services for users and system administrators.
  - Login role provides services for users to compile and launch their jobs.
- **Partition**—a group of nodes. A SLURM job cannot be scheduled to run across partitions. A root-only partition indicates that only users root or SLURM system administrator (`SlurmUser`) are allowed to allocate resource for any other user.

  Normally, one or two nodes are configured as resource manager nodes, several are configured as login nodes, the rest are compute nodes.
- **LSF partition**—a root-only partition named `lsf`, explicitly configured for the LSF system.
- **Free node**—a node that is configured in an LSF partition and is not allocated to any job
- **Available or Usable node**—a node in IDLE, ALLOCATED, COMPLETING, or DRAINING status:
  - ALLOCATED—the node has been allocated to a job.

  - COMPLETING—the node has been allocated a job that is in the process of completing. The node state is removed when all of the job processes have ended and the SLURM epilog program (if any) has ended.
  - DRAINING—the node is currently running a job, but will not be allocated to additional jobs. The node state changes to state DRAINED when the last job on it completes.
  - IDLE—the node is not allocated to any job and is available for use.
- **Unavailable node**—a node in DOWN, DRAINED, or UNKNOWN status:
  - DOWN—the node is unavailable for use.
  - DRAINED—the node is unavailable for use per system administrator request.
  - UNKNOWN—the SLURM controller has just started, and node state has not yet been determined.
- **SLURM allocation**—a set of compute nodes available for running work; same as a SLURM job. Allocations can be exclusive or shared, LSF always uses shared mode
- **SLURM job ID**—a 32-bit integer that uniquely identifies a SLURM allocation in the system. This ID can be reused.

LSF job terminology

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

  Interactive batch jobs (`bsub -I`), are started on the resource manager node.

  Normal batch jobs (`bsub` without `-I`) are started on the first node of the SLURM allocation.

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

  The `bsub` command stops display of output from the shell until the job completes, and no mail is sent to you by default. Use Ctrl-C at any time to terminate the job.
- **Serial job**—a job that requests only one slot and does not specify any of the following constraints: `mem`, `tmp`, `mincpus`, `nodes`. Serial jobs are allocated a single CPU on a shared node with minimal capacities that satisfies other allocation criteria.

  LSF always tries to run multiple serial jobs on the same node, one CPU per job.

  Parallel jobs and serial jobs cannot run on the same node.
- **Pseudo-parallel job**—a job that requests only one slot but specifies any of the following constraints: `mem`, `tmp`, `nodes` = 1, `mincpus` > 1. Pseudo-parallel jobs are allocated one node for their exclusive use.

Do NOT rely on this feature to provide node-level allocation for small jobs in job scripts. Use the SLURM[*nodes*] specification instead, along with mem, tmp, mincpus allocation options.

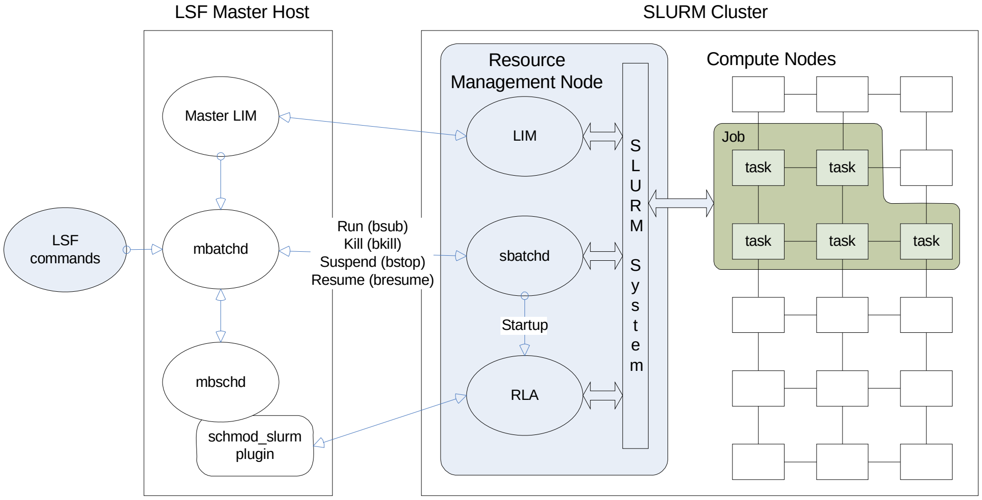
- **Parallel job**—a job that requests more than one slot, regardless of any other constraints. Parallel jobs are allocated up to the maximum number of nodes specified by:
  - SLURM[nodes=*min-max*] (if specified)

  - SLURM[nodelist=*node_list*] (if specified)

  Parallel jobs and serial jobs cannot run on the same node.

- **Small job**—a parallel job that can potentially fit into a single node of the machine, and does not explicitly request more than one node (SLURM[*nodes*] or SLURM[*node_list*] specification). LSF tries to allocate small jobs to a single node.
- **Node-level allocation**—all pseudo-parallel and parallel jobs get nodes for their exclusive use, even if the requested number of job slots is less than the total number of CPUs on those nodes. LSF provides node-level scheduling for parallel jobs and CPU-level scheduling for serial jobs. Under node-level allocation, the number of actually allocated CPUs may be greater than the requested slots.
- **First-fit allocation**: for a parallel, or pseudo-parallel job, an allocation is made left to right, for the serial job, right to left, all other criteria being equal.
- **Best-fit allocation**: all other criteria being satisfied, for parallel jobs only, the nodes with maximum number of cpus are chosen first. For both parallel and serial jobs, the nodes with minimal memory, minimal `tmp` space, and minimal weight are chosen.

Platform LSF for SLURM system architecture



What Platform LSF for SLURM does

LSF acts primarily as the workload scheduler and node allocator on top of the SLURM system, providing policy and topology-based scheduling for user tasks. SLURM provides a job execution and monitoring layer for LSF. LSF uses SLURM interfaces to:

- Query system topology information
- Make scheduling decisions
- Create allocations

LSF daemons run on a single front-end node with resource manager role, which represents the whole SLURM cluster. From the point of view of users, a SLURM cluster is one LSF host with multiple CPUs. LIM communicates with the SLURM system to get all available resource metrics for each compute node and reports resource and load information to the master LIM.

Supported features

Platform LSF for SLURM provides the following capabilities:

- As single LSF host image, LSF daemons will collect and summarize static resource metrics from compute nodes. See "How LSF reports resource metrics" on page 20 for more information.
- SLURM job submission, using the `SLURM[]` external scheduler parameter at the job-level (`bsub -ext`) and queue-level (MANDATORY_EXTSCHED or DEFAULT_EXTSCHED) to
    - Allocate nodes for a SLURM job
    - Start the job on allocated nodes
    - Support LSF job query and control.

  See "Submitting and Monitoring Jobs" on page 25 for more information.
- Topology-aware scheduling, supporting all node allocation options supported by the SLURM `srun -A` command. See "Supported srun -A allocation shape options" on page 19 for more information.
- Improved job resource accounting in SLURM—job resource usage reported accurately by `bacct`.

About this document

This document describes how to install and configure Platform LSF for SLURM.

Who should use this document

This document is intended for experienced Linux users who run applications developed by others, and for experienced system or application developers who develop, build, and run applications on a SLURM cluster.

What you should know

This document assumes you are a Linux system administrator who has experience with:

- Common system administration tasks such as working with compressed tar archives, creating user accounts, sharing and mounting shared file systems (for example, Network File System (NFS) partitions), and backing up the system
- Installing and configuring software on systems running Linux and SLURM
- Multiprocessor systems and the Message Passing Interface (MPI)
- Basic SLURM concepts (for a brief overview, see `http://www.llnl.gov/linux/slurm/overview.html`)
- Basic LSF concepts such as clusters, jobs, resources, servers, and hosts

For more information about Platform LSF

- See *Running Jobs with Platform LSF* for information about basic Platform LSF concepts
- See *Administering Platform LSF* for information about managing Platform LSF clusters
- See the *Platform LSF Command Reference* for information about Platform LSF commands
- See the *Platform LSF Configuration Reference* for information about Platform LSF features, configuration files, and environment variables

Assumptions and limitations

- A single parallel LSF job must run within a single SLURM partition. A SLURM job cannot be scheduled to run across partitions
- A shared file system is required for failover between SLURM resource manager nodes, in order to replay the event file after LSF daemons are restarted.
- Only application-level checkpointing/restart is supported.
- User-level checkpointing is not supported.
- Kernel-level checkpointing is not available.
- LSF cannot collect `maxswap` and `ndisks` static resources from compute nodes. The number of login users (`ls`) is the only load index that LSF reports. For load indices that cannot be calculated (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `it`, `tmp`, `swp`, and `mem`), `lshosts` and `lsload` displays not available (`-`).
- Except for wall-clock runtime, `bjobs` will not display job runtime resource usage
- LSF runtime resource usage limits are not enforced.
- LSF reports job accounting information only for wall-clock run time, and total number of CPUs. LSF cannot report any other job runtime resource usage information.
- LSF passes all signals for a running job to SLURM, which handles only the following signals: HUP, INT, QUIT, ABRT, KILL, ALRM, TERM, USR1, USR2, CONT, STOP, TSTP, TTIN, TTOU.
- Because SLURM does not support jobs in PTY mode on compute nodes, do not use the `bsub -Is` or `bsub -Ip` options. Interactive jobs in PTY mode are accepted on nodes with resource manager role.
- Under node-level allocation, the number of actually allocated CPUs may be greater than the requested slots. The fairshare formula is still based on slots, not on CPUs.
- Contiguous allocation will work well only if all nodes in an LSF partition of a SLURM machine are contiguous. Otherwise, LSF may reserve the wrong contiguous nodes and job will never have a chance to run.
- When LSF selects SLURM jobs to preempt, jobs to be preempted are selected from the list of preemptable candidates based on the topology-aware allocation algorithm.

  Some specialized preemption preferences, such as MINI_JOB and LEAST_RUN_TIME in the PREEMPT_FOR parameter in `lsb.params`, and others are ignored when slot preemption is required.
- LSF takes advantage of SLURM node share feature to support preemptive scheduling. When a low priority job is preempted, the job processes are suspended on the allocated nodes, and LSF places the high priority job on the same nodes. After the high priority job finishes, LSF resumes the suspended low priority job.

  Multiple jobs can be allocated to use the same node, but at any time, only one job is actually running on the node, others are suspended.
- Preemptable queue preference is not supported.
- Jobs submitted to a chunk job queue are not chunked together, but run as a normal LSF job.
- User-level account mapping is not supported.

- Job start time is not accurately predicted for resource reserving jobs with special topological requirements. The forecast time shown by `bjobs -l` is optimistic. LSF may incorrectly indicate that the job can start at a certain time, when it actually cannot start until some time after the indicated time.
- The administrator must use `brun -c` to force a job to run on a SLURM host. If the SLURM allocation cannot be satisfied for any reason, the job will be dispatched, but will be requeued and returned to pending state. The administrator can use `brun -c` again to start the job.
- By default, `brun` ignores topology options. If you specify LSF_HPC_EXTENSION="BRUN_WITH_TOPOLOGY" in `lsf.conf`, `brun` tries to run the job with the requested topology. If a topology request can be satisfied for a `brun` job, `brun` preserves the topology request. LSF allocates the resource according to the request and tries to run the job with the requested topology. If allocation fails because of topology request cannot be satisfied, job is requeued and returned to pending state.
- In MultiCluster lease model, you should export the entire SLURM cluster.
- Node names in a SLURM host must end with a number, for example `hostA1`, `hostA2`, etc. The SLURM host name itself can contain number characters, but it must begin and end with an alphabetic character. For example, `2hostA` and `hostA2` are not correct, but `host2A` is correct, and the nodes in `host2A` will be named like `host2A12`, `host2A13`, `host2A14`, etc. Note that node numbering does not necessarily start with 1.

Where to go next

- Continue to "Installing a New Platform LSF for SLURM Cluster" on page 8.

Installing a New Platform LSF for SLURM Cluster

Contents

- "Platform LSF for SLURM distribution"
- "Installing Platform LSF for SLURM (lsfinstall)"

Platform LSF for SLURM distribution

The Platform LSF for SLURM distribution consists of the following files:

- `lsf7Update3_lsfinstall.tar.Z`
- `lsf7Update3_linux2.6-glibc2.3-x86_64-slurm.tar.Z`
- `lsf7Update3_linux2.6-glibc2.3-ia64-slurm.tar.Z`
- `readme_lsf_slurm.pdf` (this document)

Installing Platform LSF for SLURM (lsfinstall)

The installation program for Platform LSF is `lsfinstall`.

What lsfinstall does

- Installs Platform LSF binary and configuration files
- Adds the Platform_HPC feature name to the PRODUCTS line of `lsf.cluster.`*`cluster_name`* if it is not already there
- Defines the following shared resources required by LSF in `lsf.shared`:

```
Begin Resource
RESOURCENAME      TYPE      INTERVAL  INCREASING   DESCRIPTION
...
   slurm          Boolean   ()        ()           (SLURM)
...
End Resource
```

- Sets maximum job slot limit to the number of CPUs that LIM reports. This is specified by MXJ=! for host type and SLINUX64 in the Host section of `LSB_CONFDIR/lsb.hosts`:

```
Begin Host
HOST_NAME     MXJ     r1m       pg     ls      tmp   DISPATCH_WINDOW  # Keywords
...
default        ()      ()        ()     ()      ()    ()               # Example
SLINUX64        !      ()        ()     ()      ()    ()
...
End Host
```

Do not change the default MXJ=! in lsb.hosts.

- Sets JOB_ACCEPT_INTERVAL=0 in `lsb.params`
- Sets the following parameters in `lsf.conf`:
  - LSF_ENABLE_EXTSCHEDULER=Y

    Enables external scheduling for Platform LSF for SLURM
  - LSB_RLA_PORT=*port_number*

    Where *port_number* is the TCP port used for communication between the Platform LSF allocation adapter (RLA) and `sbatchd` and the SLURM scheduler plugin.

    The default port number is 6883.

  - LSB_SHORT_HOSTLIST=1

    Displays an abbreviated list of hosts in `bjobs` and `bhist` for a parallel job where multiple processes of a job are running on a host. Multiple processes are displayed in the following format:

    ```
    processes*hostA
    ```

- Adds the `schmod_slurm` external scheduler plugin module name to the PluginModule section of `lsb.modules`:

  ```
  Begin PluginModule
  SCH_PLUGIN          RB_PLUGIN              SCH_DISABLE_PHASES
  schmod_default        ()                         ()
  schmod_fcfs           ()                         ()
  schmod_fairshare      ()                         ()
  schmod_limit          ()                         ()
  schmod_parallel       ()                         ()
  schmod_reserve        ()                         ()
  schmod_mc             ()                         ()
  schmod_preemption     ()                         ()
  schmod_advrsv         ()                         ()
  schmod_slurm          ()                         ()
  End PluginModule
  ```

  See the *Platform LSF Configuration Reference* for more information about `lsb.modules`.

Preinstallation checks

1. Log on as root to the node with resource manager role.
2. Check for the existence of `/var/lsf/lsfslurm`.

   If the file does not exist, touch a file with that name:

   ```
   # touch /var/lsf/lsfslurm
   ```

3. Make sure there is a shared file system available and mounted on all SLURM nodes, with a verified mount point. For example: `/hptc_cluster/lsf/tmp`.
4. Make sure that users' home directories can be accessed from all SLURM nodes.

Running lsfinstall

1. Log on as root to the node with resource manager role.
2. Change to the directory containing the distribution files.

   For example:

   ```
   # cd /tmp
   ```

3. Use the `zcat` and `tar` commands to uncompress and extract `lsf7Update3_lsfinstall.tar.Z`:

   ```
   # zcat lsf7Update3_lsfinstall.tar.Z | tar xvf -
   ```

   ***Do not* extract the Platform LSF for SLURM distribution files.**

4. Change to `lsf7_lsfinstall`:

   ```
   # cd /tmp/lsf7Update5_lsfinstall
   ```

5. Read `lsf7Update5_lsfinstall/install.config` and decide which installation variables you need to set.

6. Edit `lsf7Update5_lsfinstall/install.config` to set the installation variables you need.

   Uncomment the options you want in the template file, and replace the example values with your own settings.

   The sample values in the install.config template file are examples only. They are not default installation values.

   ---

7. Run `lsfinstall` as root:

   ```
   # ./lsfinstall -f install.config
   ```

See the *Platform LSF Command Reference* for more information about `lsfinstall` and the *Platform LSF Configuration Reference* for more information about the `install.config` file.

Required install.config variables

- LSF_TOP="/*path*"
- LSF_ADMINS="*user_name* [*user_name*...]"
- LSF_CLUSTERNAME="*cluster_name*"
- LSF_LICENSE=<none>

  `<none>` indicates that no license file is to be configured in `LSF_ENVDIR/lsf.conf` and OEM license is to be used. See "LSF Licensing" on page 21 for more information.

Variables that require an absolute path

- LSF_TOP="/*path*"
- LSF_TARDIR="/*path*"

Adding RLA port to the NIS or NIS+ database (optional)

By default, LSB_RLA_PORT is configured in `LSF_ENVDIR/lsf.conf` during installation. If you have configured other LSF ports in NIS or NIS+, you should also configure the RLA port in the NIS database *before* installing LSF. `lsfinstall` checks if this port is already defined in NIS and does not add it to `lsf.conf` if it is already defined.

See *Administering Platform LSF* for information about modifying the NIS or NIS+ database.

Where to go next

Continue to "Configuring Platform LSF for SLURM" on page 11.

Configuring Platform LSF for SLURM

Contents

- "Recommended SLURM configuration (slurm.conf)"
- "LSF configuration notes"
- "Customizing job control actions (optional)"
- "Verifying that the configuration is correct"
- "Making LSF available to users"

Recommended SLURM configuration (slurm.conf)

General parameters

- Set MaxJobCount based on number of CPUs in the SLURM cluster and the number of preemption queues you plan to use in LSF to make sure allocations are available for LSF jobs. The default value is 2000 jobs.
- Set MinJobAge to at least 1 hour to make sure LSF has enough time to get job status information after it finishes. The default value is 300 seconds. A value of zero prevents any job record purging.
- Set ReturnToService to 1, so that a DOWN node will become available for use upon registration. The default value is 0, which means that a node will remain in the DOWN state until a system administrator explicitly changes its state (even if the `slurmd` daemon registers and resumes communications).

Partitions

- Configure one partition for LSF with the name `lsf` that sets the following parameters:
  - RootOnly=YES, so that only root or the SLURM administrator can create allocations for normal user jobs. The default value is NO.
  - Shared=FORCE, so that more than one job can run on the same node. LSF uses this facility to support preemption and scheduling multiple serial jobs on the same node (node sharing). FORCE makes all nodes in the partition available for sharing without user means of disabling it. The default value is NO.
- *Do not* configure MaxNodes, MaxTime, MinNodes in an LSF partition; these parameters will conflict with LSF scheduling decisions.

LSF configuration notes

Resource to determine SLURM-enabled hosts

If not already configured, you must add the Boolean resource `slurm` in the RESOURCES column of the host section in `lsf.cluster.`*`cluster_name`* for all nodes that run in an LSF partition.

For example:

```
Begin   Host
HOSTNAME  model    type   server  r1m  mem  swp  RESOURCES    #Keywords
hostA      !        !      1       3.5  ()   ()   (slurm)
End      Host
```

The `slurm` resource is defined in the default `lsf.shared` template file at installation.

Maximum job slot limit (MXJ in lsb.hosts)

By default, the maximum job slot limit is set by `lsfinstall` to the number of CPUs that LIM reports. This is specified by MXJ=! for host type and SLINUX64 in the Host section of `LSB_CONFDIR/lsb.hosts`:

```
Begin Host
HOST_NAME      MXJ      r1m        pg     ls      tmp   DISPATCH_WINDOW   # Keywords
...
default        ()       ()         ()     ()      ()    ()                # Example
SLINUX64        !       ()         ()     ()      ()    ()
...
End Host
```

Do not change the default MXJ=! in lsb.hosts.

schmod_slurm plugin The SLURM scheduling plugin `schmod_slurm` must be configured as the last scheduler plugin module in `lsb.modules`.

Maximum number of sbatchd connections (lsb.params)

If LSF operates on a large system (for example, a system with more than 32 nodes), you may need to configure the parameter MAX_SBD_CONNS in `lsb.params`. MAX_SBD_CONNS controls the maximum number of files `mbatchd` can have open and connected to `sbatchd`. The default value of MAX_SBD_CONNS is 32.

In a very busy cluster with many jobs being dispatched, running, finishing at the same time, you may see it takes a very long time for `mbatchd` to update the status change of a job, and to dispatch new jobs. If your cluster shows this behavior, you should set MAX_SBD_CONNS = (*number of nodes*) * 2 or 300, which ever is less. Setting MAX_SBD_CONNS too high may slow down the speed of `mbatchd` dispatching new jobs.

RLA status file directory (lsf.conf) Use LSB_RLA_WORKDIR=*directory* to specify the location of the RLA status file. The RLA status file keeps track of job information to allow RLA to recover its original state when it restarts. When RLA first starts, it creates the directory defined by LSB_RLA_WORKDIR if it does not exist, then creates subdirectories for each host.

You should avoid using `/tmp` or any other directory that is automatically cleaned up by the system. Unless your installation has restrictions on the LSB_SHAREDIR directory, you should use the default:

`LSB_SHAREDIR/`*`cluster_name`*`/rla_workdir`

Other LSF configuration parameters (lsf.conf)

- LSB_RLA_PORT=*port_number*

  TCP port used for communication between the LSF HPC allocation adapter (RLA) and the SLURM scheduler plugin.

  Default: 6883

- LSB_RLA_TIMEOUT=*seconds* (optional)

  Defines the communication timeout between RLA and its clients.

  Default: 10 seconds

- LSB_RLA_UPDATE=*seconds* (optional)

  Specifies how often the LSF scheduler refreshes free node information

  Default: 600 seconds (10 minutes)
- LSB_RLA_WORKDIR=*directory* (optional)

  Directory to store RLA status file, which saves LSF job allocation information.

  Default: `LSB_SHAREDIR/`*`cluster_name`*`/rla_workdir`
- LSB_SLURM_BESTFIT=Y (optional)

  Enables best-fit node allocation.

  By default, LSF applies a first-fit allocation policy. In a heterogeneous SLURM cluster, a best-fit allocation may be preferable for clusters where a mix of serial and parallel jobs run.
- LSF_ENABLE_EXTSCHEDULER=Y

  Enables external scheduling for Platform LSF for SLURM.
- LSF_HPC_EXTENSIONS="*extension_name* ..." (optional)

  The following extensions are supported:
  - SHORT_EVENTFILE
  - SHORT_PIDLIST
  - RESERVE_BY_STARTTIME
  - BRUN_WITH_TOPOLOGY
- LSF_NON_PRIVILEGED_PORTS = Y (optional)

  Disables privileged ports usage of LSF commands and daemons, which covers functionality of LSF_MC_NON_PRIVILEGED_PORTS.
- LSF_SLURM_BINDIR=*path*

  Specifies an absolute path to the directory containing the SLURM commands. If you install SLURM in a different location from the default, you must define LSF_SLURM_BINDIR.

  Default: `/opt/hptc/slurm/bin`
- LSF_SLURM_DISABLE_CLEANUP=Y (optional)

  Disables cleanup of non-LSF jobs running in a SLURM LSF partition.

  By default, only LSF jobs are allowed to run within a SLURM LSF partition. LSF periodically cleans up any jobs submitted outside of LSF. This clean up period is defined through LSB_RLA_UPDATE.
- LSF_SLURM_TMPDIR=*path*

  Specifies the LSF `tmp` directory for SLURM machines. The default LSF_TMPDIR `/tmp` cannot be shared across nodes, so LSF_SLURM_TMPDIR must specify a path that is accessible on all SLURM nodes.

  Default: `/hptc_cluster/lsf/tmp`

The following `lsf.conf` parameters control when LIM starts to report number of usable CPUs. They are all optional.

- LSF_HPC_NCPU_COND=and|or

  Defines how two conditions are combined. By default, the value is or.
- LSF_HPC_NCPU_THRESHOLD=*threshold*

  Defines percentage of usable CPUs in LSF partition. By default, the value is 80.
- LSF_HPC_NCPU_INCREMENT=*increment*
- LSF_HPC_NCPU_INCR_CYCLES=*incr_cycles*

Two parameters determine whether system becomes stable:

- LSF_HPC_NCPU_INCR_CYCLES defines the minimum number of consecutive number of cpus checking cycle (2 minutes per each cycle). LSF_HPC_NCPU_INCREMENT defines upper limit for the number of CPUs that are changed.

For more information

- About the `lsf.conf`, `lsb.hosts`, `lsb.params`, and `lsb.queues` files, see the *Platform LSF Configuration Reference*
- About job limits and configuring hosts and queues, see *Administering Platform LSF*

Customizing job control actions (optional)

By default, LSF carries out job control actions by sending the appropriate signal to suspend, terminate, or resume a job. If your jobs need special job control actions, change the default job control actions in your queue configuration.

JOB_CONTROLS parameter (lsb.queues)

Use the JOB_CONTROLS parameter in `lsb.queues` to configure suspend, terminate, or resume job controls for the queue:

```
JOB_CONTROLS = SUSPEND[command] |
               RESUME[command] |
               TERMINATE[command]
```

where *command* is:

- The SLURM `scancel` command to control launched tasks running on remote allocated nodes
- The `kill` command to control LSF job processes running on the resource manager node.
- Any site-specific action required for job control

See the *Platform LSF Configuration Reference* for more information about the JOB_CONTROLS parameter in `lsb.queues`.

Example job control actions

```
Begin Queue
QUEUE_NAME=slurm
...
JOB_CONTROLS = TERMINATE[/opt/scripts/act.sh;kill -s TERM
-$LSB_JOBPGIDS;scancel $SLURM_JOBID]
                SUSPEND[/opt/scripts/act.sh;kill -s STOP -$LSB_JOBPGIDS;
scancel -s STOP $SLURM_JOBID]
                RESUME[/opt/scripts/act.sh;kill -s CONT -$LSB_JOBPGIDS;
scancel -s CONT $SLURM_JOBID]
...
End Queue
```

Some environments may require a TSTP signal instead of STOP.

Verifying that the configuration is correct

1. Log on as root to the LSF master host.
2. Set your LSF user environment. For example:
   - For `csh` or `tcsh`:

     `% source /usr/share/lsf/conf/cshrc.lsf`

   - For `sh`, `ksh`, or `bash`:

     `$ . /usr/share/lsf/conf/profile.lsf`

3. Start LSF:

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

   You must be root to start LSF.

4. Test your cluster by running some basic LSF commands (e.g., `lsid` and `lshosts`).
5. Use the `lsload -l` and `bhosts -l` commands to display load information for the cluster.

Example lsload -l output

The status for all nodes should be `ok`. Hosts with the static resource `slurm` defined only report the `ls` load index. The output should look similar to the this:

```
# lsload -l
HOST_NAME  status  r15s   r1m  r15m   ut    pg    io  ls  it   tmp   swp   mem
hostA      ok         -     -     -    -     -     -   1   -     -     -     -
```

See “How LSF reports resource metrics” on page 20 for more information about how load indices displayed by `lsload`.

Example bhosts -l output

The status for all nodes should be `ok`. The output should look similar to this:

```
# bhosts -l
HOST  hostA
STATUS     CPUF  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV DISPATCH_WINDOW
ok        16.00     -      8      0      0      0      0      0      -

 CURRENT LOAD USED FOR SCHEDULING:
              r15s   r1m  r15m    ut    pg    io   ls    it   tmp   swp   mem
 Total         -     -     -      -     -     -     1    -     -     -     -
 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    -     -     -     -       -     -    -     -     -      -      -
```

When a partition is down, `bhosts` shows all LSF hosts belonging to the partition as `closed_Adm`.

Making LSF available to users

After verifying that LSF is operating properly, make LSF available to your users by having them include `LSF_ENVDIR/cshrc.lsf` or `LSF_ENVDIR/profile.lsf` in their `.cshrc` or `.profile`.

Operating Platform LSF for SLURM

Contents

- "Platform LSF SLURM allocation plugin"
- "Job lifecycle"
- "Supported srun -A allocation shape options"
- "How LSF reports resource metrics"
- "LSF Licensing"
- "Best-fit and first-fit cluster-wide allocation policies"
- "Node failover"
- "Threshold conditions to report number of CPUs"
- "Running pre-execution programs"
- "Support for SLURM batch mode (srun -b)"
- "Application-level checkpointing"

Platform LSF SLURM allocation plugin

The Platform LSF external scheduler plugin for SLURM (`schmod_slurm`) is loaded on the LSF master host by `mbschd` and handles all communication between the LSF scheduler and SLURM. It translates LSF concepts (hosts and job slots) into SLURM concepts (nodes, allocation options, and allocation shape).

Platform LSF allocation adapter (RLA)

The Platform LSF allocation adapter (RLA) is located on each LSF host. RLA is started by `sbatchd` and runs on the SLURM node with resource manager role. It is the interface for the LSF SLURM plugin and the SLURM system.

To schedule a job, the SLURM external scheduler plugin calls RLA to:

- Query SLURM allocation information
- Allocate and deallocate nodes with the specified shape

The SLURM allocation plugin works with RLA to do the allocation calculation and use RLA services to allocate nodes and de-allocate nodes. `sbatchd` places jobs within allocated nodes.

Job lifecycle

How jobs run

LSF schedules jobs based on their resource requirements and communicates with the SLURM system to allocate the resources needed for the job to run. LSF provides node-level scheduling for parallel jobs and CPU-level scheduling for serial jobs.

After the LSF scheduler creates SLURM resources, it saves allocation information into LSF event file (`lsb.events`) and account file (`lsb.acct`).

When LSF starts job, it sets SLURM_JOBID and SLURM_NPROCS environment variables in the job environment. SLURM_JOBID associates the LSF job with SLURM allocated resources. SLURM_NPROCS corresponds to the `bsub -n` option. The LSF job file is started on the same node where the LSF daemons run. You must use `srun` or `mpirun` to launch real tasks on the allocated nodes.

After the job finishes, LSF cleans up the SLURM resources.

1. Job submission

Use `bsub` with the `-ext SLURM[]` external scheduler parameter to submit jobs.

In a mixed cluster, use `-R "select[defined(slurm)]"` to explicitly run jobs on a SLURM cluster.

Use `srun` to launch real parallel tasks on the allocated nodes.

2. Job scheduling

For each job, the SLURM scheduler plugin

- Merges job-level external scheduler parameter and queue-level parameters
- Loads the topology map and restores SLURM allocation for LSF jobs from RLA during the first scheduling session
- Updates the SLURM cluster free map every LSB_RLA_UPDATE interval by default or on demand if system free map is change
- Splits SLURM hosts into different host groups, one host per group to prevent jobs running across hosts
- Schedules the job based on final topology request
- Contacts RLA to create a SLURM allocation for job submission user, so that only the job owner can use the allocation
- Attaches SLURM allocation information to the job additional information string

3. Job execution

When a job starts, `sbatchd`

- Sets the SLURM_JOBID environment variable in the job control environment based on SLURM allocation information
- For `brun` jobs, `sbatchd` contacts RLA to get the job allocation string. If `brun` fails, the job is requeued
- Sets environment variables before starting up RES:
  - SLURM_NPROCS—corresponds to `bsub -n` option
  - LSF_SLURM_INPUTFILE—corresponds to `bsub -i` option
  - LSF_SLURM_OUTPUTFILE—corresponds to `bsub -o` option
  - LSF_SLURM_ERRORFILE—corresponds to `bsub -e` option

By default, pre-execution programs start on the resource manager node. You can use `srun` to launch pre-execution programs on all allocated nodes. See “Running pre-execution programs” on page 23 for more information.

Interactive batch jobs (`bsub -I`), are started on the resource manager node

Normal batch jobs (`bsub` without `-I`) are started on the first node of the SLURM allocation.

Except for sending signals to job processes running on the resource manager node, when `sbatchd` receives a signal request, it uses the SLURM `scancel` command to propagate signals to all remote tasks.

4. Job finish (Done/Exit)

For interactive jobs, `sbatchd` considers a job finished if:

- Job processes running on resource manager node are gone
  OR
- `sbatchd` and `res` are gone, and job step launched for the job does not exist any more

`sbatchd` uses the SLURM `scontrol` command to check whether job exits because of SLURM NODE FAIL. If so, `sbatchd` sets TERM_SLURM job terminate reason and job exit code as 123. Configure REQUEUE_EXIT_VALUE in the queue to enable automatic job requeue.

Post-execution commands run on the resource manager node.

After `mbschd` receives a job finish event, SLURM plugin contacts RLA to clean up SLURM job allocation.

Supported srun -A allocation shape options

SRUN option	Description	LSF equivalent
-n, --ntasks=ntasks	Number of processes (tasks) to run. Total CPUs required = ncpus * ntasks	bsub -n
-c, --cpus-per-task=ncpus	Number of CPUs per task. Minimum CPUs per node = MAX(ncpus, mincpus)	Not provided—the meaning of this option is already covered by bsub -n and -ext "SLURM[mincpus=*num_cpus*]"
-N, --nodes=min[-max]	Minimum number of nodes in the allocation request. Optionally, specifies a range of minimum to maximum nodes in the allocation. The allocation will contain at least the minimum number of nodes, but cannot exceed the maximum number of nodes.	-ext "SLURM[nodes=*min*[-*max*]]"
--mincpus=n	Minimum number of CPUs on the node. Min CPUs per node = MAX(-c ncpus, --mincpus=n) The default is 1.	-ext "SLURM[mincpus=*num_cpus*]"
--mem=MB	Minimum amount of real memory on each node, in MB	-ext "SLURM[mem=MB]"
--tmp=MB	Minimum amount of space on `/tmp` file system on each node, in MB	-ext "SLURM[tmp=MB]"
-C, --constraint=list	A list of constraints on the node allocation. The constraint list is a logical expression containing multiple features separated by | (OR — all nodes have must have at least one of the listed features) and & (AND — all nodes must have all listed features).	-ext "SLURM[constraint=*list*]"

SRUN option	Description	LSF equivalent
-w, --nodelist=node1,..,nodeN	Request a specific list of nodes. Specify a comma-separated list of nodes, or a range of nodes. The allocation will contain at least the minimum number of nodes, but cannot exceed the maximum number of nodes. nodelist=[node_list] Comma-separated list of node names or a list of node ranges that must be included in the allocation. If you specify node list with contiguous allocation, the nodes in the node list must be contiguous for the job to run. You cannot specify a non-contiguous node list. nodelist cannot specify the first execution node; SLURM starts the job on the leftmost node in the allocation.	-ext "SLURM[nodelist=*node_list*]"
-x, --exclude=node1,.. nodeN	Comma-separated list of node name ranges that must be excluded from the allocation	-ext "SLURM[exclude=*node_list*]"
-p, --partition=partition	Request resources from specified partition	One RootOnly partition for LSF named `lsf`
--contiguous	Fit the allocation in a single block of nodes with consecutive node indices	-ext "SLURM[contiguous=yes]"

How LSF reports resource metrics

LSF treats an entire SLURM cluster as one LSF host with multiple CPUs and provides a single system image end users. The following tables summarize the static resource metrics and load indices reported for SLURM clusters.

Static resource metrics

Only the following static resource metrics are available for each compute node:

- `ncpus`
- `maxmem`
- `maxtmp`

Static Resource	Description	How to Calculate
`ncpus`	Total number of usable CPUs on host	Minimum value between CPUs of all available nodes in LSF partition and number of licensed CPUs. If total number of usable CPUs is 0, LIM sets `ncpus` to 1 and close host.
`maxmem`	Maximum amount of memory available for user processes	Calculated as a minimal value over all compute nodes
`maxtmp`	The maximum `/tmp` space available on the host	Calculated as a minimal value over all compute nodes
`maxswap`	The total available swap space	Not available (–)
`ndisks`	Number of disks attached to host	Not available (–)

Load indices

The number of login users (`ls`) is the only load index that LSF reports. For load indices that cannot be calculated (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `it`, `tmp`, `swp`, and `mem`), `lshosts` and `lsload` displays not available (`-`).

Load Index	Description	How to Calculate
`r15s`	15-second exponentially averaged CPU run queue length	Not available (–)
`r1m`	1-minute exponentially averaged CPU run queue length	Not available (–)
`r15m`	15-minutes exponentially averaged CPU run queue length	Not available (–)
`ut`	CPU utilization exponentially averaged over last minute, in 0-1 interval	Not available (–)
`pg`	Memory paging rate exponentially averaged over the last minute, pages per second	Not available (–)
`io`	I/O rate exponentially averaged over the last minute, KB per second	Not available (–)
`ls`	Number of currently logged users	The value on resource manager node
`it`	Idle time of the host (keyboard is not touched on all login sessions), seconds	Not available (–)
`tmp`	Amount of free space on `/tmp`, MB	Not available (–)
`swp`	Amount of free swap space, MB	Not available (–)
`mem`	Amount of free memory, MB	Not available (–)

Custom load indices

You can configure LSF to report other load indices. For more information about LSF load indices, see *Administering Platform LSF*.

LSF Licensing

LSF licenses are managed by the Platform LSF licensing mechanism, which determines whether LSF is correctly licensed for the appropriate number of CPUs on the LSF host.

The LSF license is not transferable to any other hosts in the LSF cluster.

The following LSF features are enabled:

- `lsf_base`
- `lsf_float_client`
- `lsf_manager`

How to get additional LSF licenses

To get licenses for additional LSF features, contact Platform Computing at `license@platform.com`. For example, to enable Platform MultiCluster licenses in your LSF cluster, get a license key for the feature `lsf_multicluster`.

For more information about LSF features and licensing, see *Administering Platform LSF*.

Best-fit and first-fit cluster-wide allocation policies

By default, LSF applies a *first-fit* allocation policy to select from the nodes available for the job. The allocations are made left to right for all parallel jobs, and right to left for all serial jobs (all other job requirements being equal).

In a heterogeneous SLURM cluster, a *best-fit* allocation may be preferable for clusters where a mix of serial and parallel jobs run. In this context, best fit means: “the nodes that minimally satisfy the requirements.” Nodes with the maximum number of CPUs are chosen first. For parallel and serial jobs, the nodes with minimal memory, minimal `tmp` space, and minimal weight are chosen.

To enable best-fit allocation, specify LSB_SLURM_BESTFIT=Y in `lsf.conf`.

Node failover

The failover mechanism on SLURM clusters requires two nodes with resource manager role, where one node is master and another node is backup. At any time, LSF daemons should only be started and running on the master node.

When failover happens, the administrator must restart the LSF daemons on the backup node, and this node will become the new master node. LSF daemons or clients from other hosts can communicate with new LSF daemons.

Requeing exited jobs when the resource manager node fails

LSF jobs already started on the master node exit with exit code 122 if the master node goes down. To make sure that these jobs are restarted when the LSF daemons restart either on the backup node (as new master) or on the original master node, configure REQUEUE_EXIT_VALUES in `lsb.queues` to requeue the jobs automatically.

For example:

```
Begin Queue
QUEUE_NAME   = high
...
REQUEUE_EXIT_VALUES = 122
...
End Queue
```

Threshold conditions to report number of CPUs

When a SLURM system starts, some compute nodes may take some time to come up. If LSF starts to report the number of CPUs before all nodes are up, already queued smaller jobs might get started when a more optimal choice might be to start a larger job needing more CPUs.

To make sure all usable nodes are available for LSF to dispatch jobs to, use the following parameters in `lsf.conf` to control when LSF starts to report total usable CPUs on a SLURM cluster:

- LSF_HPC_NCPU_THRESHOLD=*threshold*

  The percentage of total usable CPUs in LSF partition. The default value is 80.

- LSF_HPC_NCPU_INCREMENT=*increment*

  Upper limit for number of CPUs that is changed since last checking cycle. The default value is 0.

- LSF_HPC_NCPU_INCR_CYCLES=*increment_cycles*

  Minimum number of consecutive cycles where the number of CPUs changed does not exceed LSF_HPC_NCPU_INCREMENT. The default value is 1. LSF checks total usable CPUs every 2 minutes.

- LSF_HPC_NCPU_COND= or | and

Define how any two thresholds are combined. The default is `or`.

Running pre-execution programs

Though LSF daemons only run on a SLURM node with resource manager role, batch jobs can run on any SLURM nodes with compute role that satisfy the scheduling and allocation requirements.

By default, where pre-execution commands run depends on the type of job:

- For interactive jobs, pre-execution commands run on the node where `sbatchd` runs, typically the resource manager node.
- For normal batch jobs, pre-execution commands run on the first node of the SLURM allocation.

Before starting a pre-exec program, LSF sets the SLURM_JOBID environment variable. To enable `srun` to launch pre-execution on the first allocated node and other allocated nodes, your pre-exec program should pick up the SLURM_JOBID environment variable. SLURM_JOBID gives LSF the information it needs to run the job on the nodes required by your pre-exec program.

Controlling where a pre-execution program starts and runs

To run a pre-execution program on

- On the resource manager node. This is the default behavior. Run the pre-execution program normally. Your pre-exec does not need to make use of SLURM_JOBID.
- On the first allocated node. Use the `srun -N 1` option. For example:

  ```
  slurm_installation_dir/bin/srun -N 1 my_pre_exec
  ```

- On all allocated nodes. Use `srun` directly with no node options. For example

  ```
  slurm_installation_dir/bin/srun my_pre_exec
  ```

See the SLURM `srun` command man page for more information about the SLURM_JOBID environment variable and the `-N` option.

Support for SLURM batch mode (srun -b)

Platform LSF for SLURM uses the SLURM `srun -b --jobid=SLURM_JOBID` command to launch jobs on the first node of the SLURM allocation.

In LSF job pre-execution programs

Do not use `srun -b --jobid=SLURM_JOBID` inside pre-execution programs. The `srun -b --jobid=SLURM_JOBID` command returns immediately after a SLURM batch job gets submitted. This can cause the pre-exec script to exit with success while the real task is still running in batch mode.

In LSF job commands

- Unless special handling provided in job command, LSF will only catch the exit code of SLURM job step that is associated with LSF job command. LSF will not catch the exit status of the job steps generated by the `srun -b` commands included in the job command.
- LSF handles the I/O for the LSF job command by using `srun` I/O options. However, subsequent `srun -b --jobid=SLURM_JOBID` do not inherit the I/O settings of the LSF job command. You must set I/O for the `srun -b --jobid=SLURM_JOBID` command included in LSF job command.

  When the `-i`, `-o`, or `-e` options to `bsub` are used, the following environment variables are set with proper values.

  - LSB_SLURM_INPUTFILE
  - LSB_SLURM_OUTPUTFILE
  - LSB_SLURM_ERRORFILE

  You must make sure your pre-execution programs and job scripts make use of these variables.

Application-level checkpointing

To enable application-level checkpoint, the checkpoint directory specified for checkpointable jobs (CHKPNT=*chkpnt_dir* parameter in the configuration of the preemptable queue) must be accessible by all SLURM nodes configured in the LSF partition.

Platform LSF creates checkpoint trigger files in the job working directory to trigger the checkpoint process of applications. Since the specified checkpoint directory must be accessible by all the nodes of LSF partition, the checkpoint trigger files will be readable by the application at run time.

For more information

About checkpointing and restart, and checkpointing specific applications, see *Administering Platform LSF*

Submitting and Monitoring Jobs

Contents

- "bsub command"
- "Running jobs on any host type"
- "Viewing nodes allocated to your job"
- "Example job submissions"

bsub command

To submit a job, use the `bsub` command.

Syntax

bsub -n *min_cpus*[,*max_cpus*]
-ext[**sched**] **"SLURM[**[*allocation_options*][**;***allocation_options*]...**]"** *job_name*

Specify allocation options for SLURM jobs either in the `-ext` option, or with DEFAULT_EXTSCHED or MANDATORY_EXTSCHED in a queue definition in `lsb.queues`.

You can abbreviate the -extsched option to -ext.

The options set by `-ext` can be combined with the queue-level MANDATORY_EXTSCHED or DEFAULT_EXTSCHED parameters.

The `-ext "SLURM[]"` options override the DEFAULT_EXTSCHED parameter, and the MANDATORY_EXTSCHED parameter overrides `-ext "SLURM[]"` options.

Controlling and monitoring jobs

Use `bkill`, `bstop`, and `bresume` to kill, suspend, and resume jobs.

Use `bjobs`, `bacct`, and `bhist` to view allocation and job resource usage information.

Examples

- Request 4 nodes, each node must have 2 cpus, 300 MB memory.

```
% bsub -n 8 -ext "SLURM[nodes=4;mincpus=2;mem=300]" my_job
```

- Request at least the following six nodes: `hostA1`, `hostA40`, `hostA41`, `hostA43`, `hostA44`, and `hostA50`, but does not want to use following two nodes: `hostA45` and `hostA46`.s

```
% bsub -n 12 -ext "SLURM[nodelist=hostA1,hostA[40-41,43-44,50];exclude=hostA[45-46]]
my_job
```

- Request at least 5 and at most 8 contiguous nodes.

```
% bsub -n 16 -ext "SLURM[nodes=5-8;contiguous=yes]" my_job
```

- Request exactly 5 nodes.

```
% bsub -n 10 -ext "SLURM[nodes=5-5]" my_job
```

- Submit single `srun` job:

```
% bsub -n 4 -ext "SLURM[nodes=2]" srun srun_options my_job
```

- Submit single MPI job:

```
% bsub -n 8 -ext "SLURM[nodes=2]" mpirun -srun
srun_options myjob
```

- Submit a job script:

```
% bsub -n 8 -ext "SLURM[nodes=2]" < jobscript
```

For more information

- About `-ext` options, see “bsub command” on page 35
- About MANDATORY_EXTSCHED and DEFAULT_EXTSCHED, see “lsb.queues file” on page 37
- About job operations, see *Administering Platform LSF*
- About `bsub`, see the *Platform LSF Command Reference*

Running jobs on any host type

You can specify several types of allocation options at job submission and LSF will schedule jobs appropriately. Jobs that do not specify SLURM-related options can be dispatched to SLURM hosts, and jobs with SLURM-related options can be dispatched to non-SLURM hosts.

Use the LSF resource requirements specification (`-R` option of `bsub` or RES_REQ in queue definition in `lsb.queues`) to identify the host types required for your job.

SLURM hosts can exist in the same LSF cluster with other host types. Use the `-R` option to define host type resource requirements. For example. 64-bit Linux hosts are host type LINUX64 and SLURM hosts are host type `SLINUX64`.

Examples

- Run `myjob` on a 64-bit Linux host or a SLURM host if one is available, but *not* both:

```
% bsub -n 8 -R "LINUX64 || SLINUX64" -ext "SLURM[nodes=4-4]" myjob
```

  If `myjob` runs on a SLURM host, the `SLURM[nodes=4-4]` allocation option is applied. If it runs on a 64-bit Linux host, the SLURM option is ignored.

- Run `myjob` on any host type, and apply allocation options appropriately:

```
% bsub-n 8 -R "type==any" -ext "SLURM[nodes=4-4];RMS[ptile=2]" myjob
```

  If `myjob` runs on an RMS-enabled host, the RMS `ptile` option is applied. If it runs on any other host type, the SLURM and RMS options are ignored.

Viewing nodes allocated to your job

After LSF allocates nodes for job, it attaches allocation information to job, so you can view allocation through `bjobs`, `bhist`, and `bacct`.

The job allocation information string has the form:

`slurm_id=`*`slurm_jobid`***`;ncpus=`***`number`***`;slurm_alloc=`***`node_list`***`;`**

`Where:`

- `slurm_id`—SLURM allocation ID (SLURM_JOBID environement variable).
- `ncpus`—actual total number of CPUs allocated to job. Because LSF uses node-level allocation for parallel jobs, the `ncpus` value may be larger than the number of CPUs requested by the `bsub -n` option at job submission.
- `slurm_alloc`—allocated node list.

For example:

```
Tue Aug 31 16:22:27: slurm_id=60;ncpus=4;slurm_alloc=n[14-15];
```

Running jobs (bjobs -l)

Use `bjobs -l` to see SLURM allocation information for a running job.

For example, the following job allocates nodes on hosts `hostA`:

```
% bsub -n 8 -ext"SLURM[]" mpirun -srun /usr/share/lsf7slurm/bin/hw
Job <7267> is submitted to default queue <normal>.
```

`bjobs` output looks like this:

```
% bjobs -l 7267

Job <7267>, User <user1>, Project <default>, Status <DONE>, Queue <normal>,
                     Extsched <SLURM[]>, Command <mpirun -srun /usr/share/lsf7
                     slurm/bin/hw>
Thu Sep 16 15:29:06: Submitted from host <hostA>, CWD </usr/share/lsf7slurm/b
                     in>, 8 Processors Requested;
Thu Sep 16 15:29:16: Started on 8 Hosts/Processors <8*hostA>;
Thu Sep 16 15:29:16: slurm_id=21795;ncpus=8;slurm_alloc=n[13-16];
Thu Sep 16 15:29:23: Done successfully. The CPU time used is 0.0 seconds.

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

 EXTERNAL MESSAGES:
 MSG_ID FROM       POST_TIME      MESSAGE                            ATTACHMENT
 0          -             -                        -                      -
 1      user1      Sep 16 15:29   SLURM[]                                 N
```

Finished jobs (bhist -l)

Use `bhist -l` to see SLURM allocation information for finished jobs. For example:

```
% bhist -l 7267

Job <7267>, User <user1>, Project <default>, Extsched <SLURM[]>, Command <mpi
                     run -srun /usr/share/lsf7slurm/bin/hw>
Thu Sep 16 15:29:06: Submitted from host <hostA>, to Queue <normal>, CWD </u
                     sr/share/lsf7slurm/bin>, 8 Processors Requested;
Thu Sep 16 15:29:16: Dispatched to 8 Hosts/Processors <8*hostA>;
Thu Sep 16 15:29:16: slurm_id=21795;ncpus=8;slurm_alloc=n[13-16];
Thu Sep 16 15:29:16: Starting (Pid 5804);
Thu Sep 16 15:29:23: Done successfully. The CPU time used is 0.0 seconds;
Thu Sep 16 15:29:23: Post job process done successfully;

Summary of time in seconds spent in various states by  Thu Sep 16 15:29:23
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
  10       0        7        0        0        0        17
```

Job accounting information (bacct -l)

Use `bacct -l` to see SLURM allocation information logged to `lsb.acct`. For example:

```
% bacct -l 7267

Accounting information about jobs that are:
  - submitted by all users.
  - accounted on all projects.
  - completed normally or exited
  - executed on all hosts.
  - submitted to all queues.
  - accounted on all service classes.
------------------------------------------------------------------------------

Job <7267>, User <user1>, Project <default>, Status <DONE>, Queue <normal>,
                     Command <mpirun -srun /usr/share/lsf7slurm/bin/hw>
Thu Sep 16 15:29:06: Submitted from host <hostA>, CWD </usr/share/lsf7slurm/b
                     in>;
Thu Sep 16 15:29:16: Dispatched to 8 Hosts/Processors <8*hostA>;
Thu Sep 16 15:29:16: slurm_id=21795;ncpus=8;slurm_alloc=n[13-16];
Thu Sep 16 15:29:23: Completed <done>.

Accounting information about this job:
     Share group charged </user1>
     CPU_T     WAIT     TURNAROUND   STATUS     HOG_FACTOR    MEM    SWAP
      0.05       10             17     done         0.0029     0K      0K
------------------------------------------------------------------------------

SUMMARY:      ( time unit: second )
 Total number of done jobs:       1      Total number of exited jobs:     0
 Total CPU time consumed:       0.1      Average CPU time consumed:     0.1
 Maximum CPU time of a job:     0.1      Minimum CPU time of a job:     0.1
 Total wait time in queues:     7.0
 Average wait time in queue:    7.0
 Maximum wait time in queue:    7.0      Minimum wait time in queue:    7.0
 Average turnaround time:        28 (seconds/job)
 Maximum turnaround time:        28      Minimum turnaround time:        28
 Average hog factor of a job:  0.00 ( cpu time / turnaround time )
 Maximum hog factor of a job:  0.00      Minimum hog factor of a job:  0.00
```

Example job submissions

Environment On one SLURM cluster, the `lsf` partition configures 4 compute nodes, each with 2 CPUs. LSF is installed on `hostA`. `lsid`, `lshosts`, `bhosts`, and `sinfo` show the configuration:

```
% lsid
Platform LSF HPC 7.0.3.108010 for SLURM, Jun  3 2008
Copyright 1992-2009 Platform Computing Corporation

My cluster name is cluster1
My master name is hostA

% lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
hostA       SLINUX6 Itanium2  16.0     8     1M      -    Yes (slurm)

% bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
hostA              ok              -      8      0      0      0      0      0
% sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4  alloc n[13-16]
```

Submit a job script with -I

Multiple `srun` commands are specified inside the script:

```
% cat myjobscript.sh
#!/bin/sh
srun hostname
srun uname -a
```

Submit the job:

```
% bsub -I -n 4 < myjobscript.sh
Job <1> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on hostA>>
n13
n13
n14
n14
Linux n13 2.4.21-15.9smp #1 SMP Wed Aug 25 01:07:12 EDT 2009 ia64 ia64 ia64
GNU/Linux
Linux n13 2.4.21-15.9smp #1 SMP Wed Aug 25 01:07:12 EDT 2009 ia64 ia64 ia64
GNU/Linux
Linux n14 2.4.21-15.9smp #1 SMP Wed Aug 25 01:07:12 EDT 2009 ia64 ia64 ia64
GNU/Linux
Linux n14 2.4.21-15.9smp #1 SMP Wed Aug 25 01:07:12 EDT 2009 ia64 ia64 ia64
GNU/Linux
```

Submit /bin/sh with -Ip

Use the SLURM `sinfo` command to show node information:

```
% sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4  alloc n[13-16]
```

Submit the job:

```
% bsub -n8 -Ip /bin/sh
Job <2> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on hostA>>
```

`sinfo` shows the allocation:

```
% sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4  alloc n[13-16]
```

View the SLURM job ID:

```
% env | grep SLURM
SLURM_JOBID=18
SLURM_NPROCS=8
```

Run some commands:

```
% srun hostname
n13
n13
n14
n14
n15
n15
n16
n16

% srun -n 5 hostname
n13
n13
n14
n15
n16

% exit
exit
```

Use `bjobs` to see the interactive jobs:

```
% bjobs -l 2

Job <2>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Int
                     eractive pseudo-terminal mode, Command </bin/sh>
Wed Sep 22 18:05:56: Submitted from host <hostA>, CWD <$HOME>, 8 Processors
                     Requested;
Wed Sep 22 18:06:06: Started on 8 Hosts/Processors <8*hostA>;
Wed Sep 22 18:06:06: slurm_id=18;ncpus=8;slurm_alloc=n[13-16];
Wed Sep 22 18:10:37: Done successfully. The CPU time used is 0.6 seconds.
```

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

Use `bhist` to see the history of the finished jobs:

```
% bhist -l 2

Job <2>, User <user1>, Project <default>, Interactive pseudo-terminal mode,
                     Command </bin/sh>
Wed Sep 22 18:05:56: Submitted from host <hostA>, to Queue <normal>, CWD <$H
                     OME>, 8 Processors Requested;
Wed Sep 22 18:06:06: Dispatched to 8 Hosts/Processors <8*hostA>;
Wed Sep 22 18:06:06: slurm_id=18;ncpus=8;slurm_alloc=n[13-16];
Wed Sep 22 18:06:06: Starting (Pid 9462);
Wed Sep 22 18:10:24: Done successfully. The CPU time used is 0.6 seconds;
Wed Sep 22 18:10:37: Post job process done successfully;

Summary of time in seconds spent in various states by  Wed Sep 22 18:10:37
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
  10       0        258      0        0        0        268
```

Use the SLURM `sinfo` command to see node state:

```
% sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4  alloc n[13-16]
```

Use the SLURM `scontrol` command to see the SLURM job information:

```
% scontrol show job 18
JobId=18 UserId=user1(502) GroupId=lsfadmin(502)
   Name=lsf7slurm@2 JobState=COMPLETED
   Priority=4294901743 Partition=lsf BatchFlag=0
   AllocNode:Sid=n16:8833 TimeLimit=UNLIMITED
   StartTime=09/22-18:06:01 EndTime=09/22-18:10:24
   NodeList=n[13-16] NodeListIndecies=-1
   ReqProcs=0 MinNodes=0 Shared=0 Contiguous=0
   MinProcs=0 MinMemory=0 Features=(null) MinTmpDisk=0
   ReqNodeList=(null) ReqNodeListIndecies=-1
   ExcNodeList=(null) ExcNodeListIndecies=-1
```

Run an MPI job

Submit the job:

```
% bsub -I -n6 -ext "SLURM[nodes=3]" /opt/mpi/bin/mpirun -srun
/usr/share/lsf7slurm/bin/hw
Job <6> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on hostA>>
I'm process 0! from ( n13 pid 27222)
Greetings from process 1! from ( n13 pid 27223)
Greetings from process 2! from ( n14 pid 14011)
Greetings from process 3! from ( n14 pid 14012)
Greetings from process 4! from ( n15 pid 18227)
Greetings from process 5! from ( n15 pid 18228)
mpirun exits with status: 0
```

Use `bjobs` to see the job:

```
% bjobs -l 6

Job <6>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Int
                     eractive mode, Extsched <SLURM[nodes=3]>, Command </opt/
                     mpi/bin/mpirun -srun /usr/share/lsf7slurm/bin/hw>
Wed Sep 22 18:16:51: Submitted from host <hostA>, CWD <$HOME>, 6 Processors
                     Requested;
Wed Sep 22 18:17:02: Started on 6 Hosts/Processors <6*hostA>;
Wed Sep 22 18:17:02: slurm_id=22;ncpus=6;slurm_alloc=n[13-15];
Wed Sep 22 18:17:09: Done successfully. The CPU time used is 0.0 seconds.

 SCHEDULI7NG PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

 EXTERNAL MESSAGES:
 MSG_ID FROM      POST_TIME      MESSAGE                             ATTACHMENT
 0      -         -              -                                        -
 1      user1     Sep 22 18:16   SLURM[nodes=3]                           N
```

Use `bhist` to see the history of the finished job:

```
% bhist -l 6

Job <6>, User <user1>, Project <default>, Interactive mode, Extsched <SLURM[
                     nodes=3]>, Command </opt/mpi/bin/mpirun -srun /usr/share
                     /lsf7slurm/bin/hw>
Wed Sep 22 18:16:51: Submitted from host <hostA>, to Queue <normal>, CWD <$H
                     OME>, 6 Processors Requested;
Wed Sep 22 18:17:02: Dispatched to 6 Hosts/Processors <6*hostA>;
Wed Sep 22 18:17:02: slurm_id=22;ncpus=6;slurm_alloc=n[13-15];
Wed Sep 22 18:17:02: Starting (Pid 11216);
Wed Sep 22 18:17:09: Done successfully. The CPU time used is 0.0 seconds;
Wed Sep 22 18:17:09: Post job process done successfully;

Summary of time in seconds spent in various states by  Wed Sep 22 18:17:09
```

```
  PEND      PSUSP     RUN       USUSP     SSUSP     UNKWN     TOTAL
  11        0         7         0         0         0         18
```

Run a job with a SLURM allocation request

Submit jobs to a SLURM cluster with three compute nodes (`n13`, `n14`, and `n16`).

- Submit a job requesting 2 slots on any 2 nodes:

```
% bsub -I -n 2 -ext "SLURM[nodes=2]" srun hostname
Job <8> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on hostA>>
n13
n14
```

Use `bjobs` to see the job:

```
% bjobs -l 8

Job <8>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Int
                     eractive mode, Extsched <SLURM[nodes=2]>, Command <srun ho
                     stname>
Wed Sep 22 18:18:58: Submitted from host <hostA>, CWD <$HOME>, 2 Processors
                     Requested;
Wed Sep 22 18:19:07: Started on 2 Hosts/Processors <2*hostA>;
Wed Sep 22 18:19:07: slurm_id=24;ncpus=4;slurm_alloc=n[13-14];
Wed Sep 22 18:19:12: Done successfully. The CPU time used is 0.0 seconds.

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

 EXTERNAL MESSAGES:
 MSG_ID FROM       POST_TIME      MESSAGE                            ATTACHMENT
 0          -            -                        -                      -
 1      user1      Sep 22 18:18   SLURM[nodes=2]                         N
```

- Submit a job requesting 2 slots on specific nodes:

```
% bsub -I -n 4 -ext "SLURM[nodelist=n[14,16]]"
srun hostname
Job <9> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on hostA>>
n14
n14
n16
n16
```

Use `bjobs` to see the job:

```
% bjobs -l 9

Job <9>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Int
                     eractive mode, Extsched <SLURM[nodelist=n[14,16]]>, Comman
                     d <srun hostname>
Wed Sep 22 18:20:00: Submitted from host <hostA>, CWD <$HOME>, 4 Processors
```

```
                     Requested;
Wed Sep 22 18:20:07: Started on 4 Hosts/Processors <4*hostA>;
Wed Sep 22 18:20:07: slurm_id=25;ncpus=4;slurm_alloc=n[14,16];
Wed Sep 22 18:20:24: Done successfully. The CPU time used is 0.0 seconds.

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

 EXTERNAL MESSAGES:
 MSG_ID FROM       POST_TIME      MESSAGE                           ATTACHMENT
 0          -             -                        -                    -
 1      user1      Sep 22 18:20   SLURM[nodelist=n[14,16]]              N
```

Command Reference

bsub command

-ext[**sched**] **"SLURM[**[*allocation_options*][**;***allocation_options*] **...]"**

Specifies allocation options for SLURM jobs.

You can abbreviate the -extsched option to -ext.

The options set by `-ext` can be combined with the queue-level MANDATORY_EXTSCHED or DEFAULT_EXTSCHED parameters.

The `-ext "SLURM[]"` options override the DEFAULT_EXTSCHED parameter, and the MANDATORY_EXTSCHED parameter overrides `-ext "SLURM[]"` options.

See "lsb.queues file" on page 37 for more information about MANDATORY_EXTSCHED and DEFAULT_EXTSCHED.

allocation_options

Specifies the SLURM allocation shape options for the job:

- **constraint=**[*constraint_list*]

  A list of constraints on the node allocation.

  The constraints are features that have been assigned to the nodes by the SLURM administrator. A feature is an arbitrary string that represents some characteristic associated with the node. There is no value associated with a feature; a node either has a feature or it does not. If desired a feature may contain a numeric component indicating, for example, processor speed. By default a node has no features.

  The constraint list is a logical expression containing multiple features separated by `|` (OR—all nodes have must have at least one of the listed features) and `&` (AND—all nodes must have all listed features).

  For example:

  ```
  constraint=bigmem|bigtmp
  ```

  If no nodes have the requested features, the job remains pending.

- **contiguous=**[**yes** | **no**]

  Fit the allocation in a single block of nodes with consecutive node indices

  If the requested block of contiguous nodes is not available for allocation, the allocation fails and the job remains pending.

  If you specify contiguous allocation with a node list, the nodes in the node list must be contiguous for the job to run. You cannot specify a non-contiguous node list.

- **exclude=**[*node_list*]

  Comma-separated list of node name ranges that must be excluded from the allocation

- **mem=**[*integer*]

  Minimum amount of real memory on each node, in MB

- **`mincpus=`**[*num_cpus*]

  Minimum number of CPUs on the node. The default is 1.
- **`nodes=`**[*min_nodes*[**`-`***max_nodes*]]

  Minimum number of nodes in the allocation request.Optionally, specifies a range of minimum to maximum nodes in the allocation. The allocation will contain at least the minimum number of nodes, but cannot exceed the maximum number of nodes.
- **`nodelist=`**[*node_list*]

  Comma-separated list of node names or a list of node ranges that must be included in the allocation.

  If you specify node list with contiguous allocation, *all* the nodes in the node list must be contiguous for the job to run. You cannot specify any other non-contiguous nodes in the node list.

  `nodelist` cannot specify the first execution node; SLURM starts the job on the leftmost node in the allocation.
- **`tmp=`**[*integer*]

  Minimum amount of space on `/tmp` file system on each node, in MB

Usage

- Option names are not case sensitive. `mem=300` is the same as `MEM=300`.
- You can specify any SLURM[] option only once.
- The `bsub -n` option cannot be less than the number of nodes specified in the `nodes` or `nodelist` options.
- The values for `mincpus`, `mem`, and `tmp` options must be either positive integers or empty.
- In a range expression for `nodes`, `nodelist` and `exclude` options, the minimum value cannot be greater than the maximum value specified.
- For `nodelist` and `exclude` options, node names in *node_list* must end with a number, for example `hostA1`, `hostA2`, etc. The SLURM host name itself can contain number characters, but it must begin and end with an alphabetic character. For example, `2hostA` and `hostA2` are not correct, but `host2A` is correct, and the nodes in `host2A` will be named like `host2A12`, `host2A13`, `host2A14`, etc. Note that node numbering does not necessarily start with 1.

If allocation options are set in DEFAULT_EXTSCHED, and you do not want to specify values for these options, use the keyword with no value in the `-ext` option of `bsub`. For example, if `DEFAULT_EXTSCHED=SLURM[nodes=2]`, and you do not want to specify any node option at all, use `-ext "SLURM[nodes=]"`.

File Reference

Contents

- "lsb.queues file"
- "lsf.conf file"

lsb.queues file

DEFAULT_EXTSCHED

Syntax **`DEFAULT_EXTSCHED=SLURM[`***[allocation_options][***`;`***allocation_options] ...]***`]`**

Description Specifies SLURM allocation options for the queue.

`-ext` options on the `bsub` command are merged with DEFAULT_EXTSCHED options, and `-ext` options override any conflicting queue-level options set by DEFAULT_EXTSCHED.

For example, if `DEFAULT_EXTSCHED=SLURM[nodes=2;tmp=100]` and a job is submitted with `-ext "SLURM[nodes=3;tmp=]"`, LSF uses the following resulting options for scheduling:

```
SLURM[nodes=3]
```

`nodes=3` in the `-ext` option overrides `nodes=2` in DEFAULT_EXTSCHED, and `tmp=` in `-ext` option overrides `tmp=100` in DEFAULT_EXTSCHED.

DEFAULT_EXTSCHED can be used in combination with MANDATORY_EXTSCHED in the same queue. For example:

- `-ext "SLURM[nodes=3;tmp=]"`
- `DEFAULT_EXTSCHED=SLURM[nodes=2;tmp=100]`
- `MANDATORY_EXTSCHED=SLURM[contiguous=yes;tmp=200]`

LSF uses the resulting options for scheduling:

```
SLURM[nodes=3;contiguous=yes;tmp=200]
```

`nodes=3` in the `-ext` option overrides `nodes=2` in DEFAULT_EXTSCHED, and `tmp=` in `-ext` option overrides `tmp=100` in DEFAULT_EXTSCHED. MANDATORY_EXTSCHED adds `contiguous=yes`, and overrides `tmp=` in `-ext` option and `tmp=100` in DEFAULT_EXTSCHED with `tmp=200`.

If allocation options are set in DEFAULT_EXTSCHED, and you do not want to specify values for these options, use the keyword with no value in the `-ext` option of `bsub`. For example, if `DEFAULT_EXTSCHED=SLURM[nodes=2]`, and you do not want to specify any node option at all, use `-ext "SLURM[nodes=]"`.

See "bsub command" on page 25 for more information.

Default Undefined.

MANDATORY_EXTSCHED

Syntax **`MANDATORY_EXTSCHED=SLURM[`***[allocation_options][***`;`***allocation_options] ...]***`]`**

Description Specifies mandatory SLURM allocation options for the queue.

`-ext` options on the `bsub` command are merged with MANDATORY_EXTSCHED options, and MANDATORY_EXTSCHED options override any conflicting job-level options set by `-ext`.

Overrides `-ext` options on the `bsub` command.

For example, if `MANDATORY_EXTSCHED=SLURM[contiguous=yes;tmp=200]` and a job is submitted with `-ext "SLURM[nodes=3;tmp=100]"`, LSF uses the following resulting options for scheduling:

```
"SLURM[nodes=3;contiguous=yes;tmp=200]"
```

MANDATORY_EXTSCHED can be used in combination with DEFAULT_EXTSCHED in the same queue. For example:

- `-ext "SLURM[nodes=3;tmp=]"`
- `DEFAULT_EXTSCHED=SLURM[nodes=2;tmp=100]`
- `MANDATORY_EXTSCHED=SLURM[contiguous=yes;tmp=200]`

LSF uses the resulting options for scheduling:

```
SLURM[nodes=3;contiguous=yes;tmp=200]
```

`nodes=3` in the `-ext` option overrides `nodes=2` in DEFAULT_EXTSCHED, and `tmp=` in `-ext` option overrides `tmp=100` in DEFAULT_EXTSCHED. MANDATORY_EXTSCHED adds `contiguous=yes`, and overrides `tmp=` in `-ext` option and `tmp=100` in DEFAULT_EXTSCHED with `tmp=200`.

If you want to prevent users from setting allocation options in the `-ext` option of `bsub`, use the keyword with no value. For example, if the job is submitted with `-ext "SLURM[nodes=4]"`, use `MANDATORY_EXTSCHED=RMS[nodes=]` to override this setting.

See “bsub command” on page 25 for more information.

Default Undefined.

lsf.conf file

LSB_RLA_PORT

Syntax **`LSB_RLA_PORT=`***port_number*

Description TCP port used for communication between the LSF HPC allocation adapter (RLA) and the SLURM scheduler plugin.

Default 6883

LSB_RLA_TIMEOUT

Syntax	**LSB_RLA_TIMEOUT=***seconds*
Description	Defines the communications timeout between RLA and its clients (e.g., `sbatchd` and SLURM scheduler plugin.)
Default	10 seconds

LSB_RLA_UPDATE

Syntax	**LSB_RLA_UPDATE=***seconds*
Description	Specifies how often the LSF scheduler refreshes free node information from RLA.
Default	600 seconds

LSB_RLA_WORKDIR

Syntax	**LSB_RLA_WORKDIR=***directory*
Description	Directory to store the RLA status file. Allows RLA to recover its original state when it restarts. When RLA first starts, it creates the directory defined by LSB_RLA_WORKDIR if it does not exist, then creates subdirectories for each host. You should avoid using `/tmp` or any other directory that is automatically cleaned up by the system. Unless your installation has restrictions on the LSB_SHAREDIR directory, you should use the default for LSB_RLA_WORKDIR.
Default	`LSB_SHAREDIR/`*`cluster_name`*`/rla_workdir`

LSB_SLURM_BESTFIT

Syntax	**LSB_SLURM_BESTFIT=y** | **Y**
Description	Enables best-fit node allocation. By default, LSF applies a *first-fit* allocation policy to select from the nodes available for the job. The allocations are made left to right for all parallel jobs, and right to left for all serial jobs (all other job requirements being equal). In a heterogeneous SLURM cluster, a *best-fit* allocation may be preferable for clusters where a mix of serial and parallel jobs run. In this context, best fit means: “the nodes that minimally satisfy the requirements.” Nodes with the maximum number of CPUs are chosen first. For parallel and serial jobs, the nodes with minimal memory, minimal `tmp` space, and minimal weight are chosen.
Default	Undefined

LSF_ENABLE_EXTSCHEDULER

Syntax	**LSF_ENABLE_EXTSCHEDULER=y** | **Y**
Description	Enables external scheduling for Platform LSF for SLURM
Default	Y (automatically set by `lsfinstall`)

LSF_HPC_EXTENSIONS

Syntax **LSF_HPC_EXTENSIONS="***extension_name ...***"**

Description Enables Platform LSF extensions.

Valid values The following extension names are supported:

- SHORT_EVENTFILE—compresses long host name lists when event records are written to `lsb.events` and `lsb.acct` for large parallel jobs. The short host string has the format:

  `number_of_hosts*real_host_name`

  When SHORT_EVENTFILE is enabled, older daemons and commands (pre-LSF Version 6) cannot recognize the lsb.acct and lsb.events file format.

  For example, if the original host list record is

  ```
  6 "hostA" "hostA" "hostA" "hostA" "hostB" "hostC"
  ```

  redundant host names are removed and the host count is changed so that the short host list record becomes

  ```
  3 "4*hostA" "hostB" "hostC"
  ```

  When LSF_HPC_EXTENSION="SHORT_EVENTFILE" is set, and LSF reads the host list from `lsb.events` or `lsb.acct`, the compressed host list is expanded into a normal host list.

  Applies to the following events:

  - JOB_START when a normal job is dispatched
  - JOB_FORCE when a job is forced with `brun`
  - JOB_CHUNK when a job is inserted into a job chunk
  - JOB_FORWARD when a job is forwarded to a MultiCluster leased host
  - JOB_FINISH in `lsb.acct`
- SHORT_PIDLIST—shortens the output from `bjobs` not to include all of the process IDs (PIDs) for a job. `bjobs` displays only the first ID and a count of the process group IDs (PGIDs) and process IDs for the job.

  Without SHORT_PIDLIST, `bjobs -l` displays all the PGIDs and PIDs for the job. With SHORT_PIDLIST set, `bjobs -l` displays a count of the PGIDS and PIDs.
- RESERVE_BY_STARTTIME—LSF selects the reservation that will give the job the earliest predicted start time.

  By default, if multiple host groups are available for reservation, LSF chooses the largest possible reservation based on number of slots. When backfill is configured, this can lead to larger jobs not running as their start times get pushed further into the future.
- BRUN_WITH_TOPOLOGY—if a topology request can be satisfied for a `brun` job, `brun` preserves the topology request. LSF allocates the resource according to the request and tries to run the job with the requested topology. If allocation fails because of topology request cannot be satisfied, job is requeued.

By default, if BRUN_WITH_TOPOLOGY is not specified, the job topology request is ignored by the scheduler when it creates an allocation.

Default Undefined

LSF_HPC_NCPU_COND

Syntax **LSF_HPC_NCPU_COND=and | or**

Description Defines how any two LSF_HPC_NCPU_* thresholds are combined.

Default `or`

LSF_HPC_NCPU_INCREMENT

Syntax **LSF_HPC_NCPU_INCREMENT=***increment*

Description Defines the upper limit for the number of CPUs that are changed since the last checking cycle.

Default 0

LSF_HPC_NCPU_INCR_CYCLES

Syntax **LSF_HPC_NCPU_INCR_CYCLES=***incr_cyscles*

Description Minimum number of consecutive cycles where the number of CPUs changed does not exceed LSF_HPC_NCPU_INCREMENT. LSF checks total usable CPUs every 2 minutes.

Default 1

LSF_HPC_NCPU_THRESHOLD

Syntax **LSF_HPC_NCPU_THRESHOLD=***threshold*

Description LSF_HPC_NCPU_THRESHOLD=threshold

The percentage of total usable CPUs in the LSF partition.

Default 80

LSF_NON_PRIVILEGED_PORTS

Syntax **LSF_NON_PRIVILEGED_PORTS=y | Y**

Description Disables privileged ports usage.

By default, LSF daemons and clients running under root account will use privileged ports to communicate with each other. Without LSF_NON_PRIVILEGED_PORTS defined, and if LSF_AUTH is not defined in `lsf.conf`, LSF daemons check privileged port of request message to do authentication.

If LSF_NON_PRIVILEGED_PORTS=Y is defined, LSF clients (LSF commands and daemons) will not use privileged ports to communicate with daemons and LSF daemons will not check privileged ports of incoming requests to do authentication.

Default Undefined

LSF_SLURM_BINDIR

Syntax **`LSF_SLURM_BINDIR=`***absolute_path*

Description Specifies an absolute path to the directory containing the SLURM commands. If you install SLURM in a different location from the default, you must define LSF_SLURM_BINDIR.

Default `/opt/hptc/slurm/bin`

LSF_SLURM_DISABLE_CLEANUP

Syntax **`LSF_SLURM_DISABLE_CLEANUP=y`** | **`Y`**

Description Disables cleanup of non-LSF jobs running in a SLURM LSF partition.

By default, only LSF jobs are allowed to run within a SLURM LSF partition. LSF periodically cleans up any jobs submitted outside of LSF. This clean up period is defined through LSB_RLA_UPDATE.

For example, the following `srun` job is not submitted through LSF, so it is terminated:

```
% srun -n 4 -p lsf sleep 100000
srun: error: n13: task[0-1]: Terminated
srun: Terminating job
```

If LSF_SLURM_DISABLE_CLEANUP=Y is set, this job would be allowed to run.

Default Undefined

LSF_SLURM_TMPDIR

Syntax **`LSF_SLURM_TMPDIR=`***path*

Description Specifies the LSF `tmp` directory for SLURM clusters. The default LSF_TMPDIR `/tmp` cannot be shared across nodes, so LSF_SLURM_TMPDIR must specify a path that is accessible on all SLURM nodes.

LSF_SLURM_TMPDIR only affects SLURM machine configuration. It is ignored on other systems in a mixed cluster environment.

The location of LSF `tmp` directory is determined in the following order:

- LSF_SLURM_TMPDIR, if defined
- LSF_TMPDIR, if defined
- The default shared directory `/hptc_cluster/lsf/tmp`

Default `/hptc_cluster/lsf/tmp`

Copyright

Document redistribution policy

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

Internal redistribution

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

Trademarks

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

ACCELERATING INTELLIGENCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOBSCHEDULER, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Corporation in the United States and in other jurisdictions.

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

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

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

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

Macrovision, Globetrotter, FLEXlm, FLEXnet, FLEXnet Manager, FLEXnet Connector, are registered trademarks or trademarks of Macrovision Corporation in the United States of America and/or other countries.

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