# HP XC System Software XC User Guide

Version 4.0

HP Part Number: A-XCUSR-40a
Published: February 2009

# Table of Contents

# List of Figures

# List of Tables

# List of Examples

# About This Document

This document provides information about using the features and functions of the HP XC System Software. It describes how the HP XC user and programming environments differ from standard Linux® system environments. In addition, this manual focuses on building and running applications in the HP XC environment and is intended to guide an application developer to take maximum advantage of HP XC features and functions by providing an understanding of the underlying mechanisms of the HP XC programming environment.

An HP XC system is integrated with several open source software components. Some open source software components are being used for underlying technology, and their deployment is transparent. Some open source software components require user-level documentation specific to HP XC systems, and that kind of information is included in this document, when required.

HP relies on the documentation provided by the open source developers to supply the information you need to use their product. For links to open source software documentation for products that are integrated with the HP XC system, see "Supplementary Software Products" (page 13).

Documentation for third-party hardware and software components that are supported on the HP XC system is supplied by the third-party vendor. However, information about the operation of third-party software is included in this document if the functionality of the third-party component differs from standard behavior when used in the XC environment. In this case, HP XC documentation supersedes information supplied by the third-party vendor. For links to related third-party Web sites, see "Supplementary Software Products" (page 13).

Standard Linux® administrative tasks or the functions provided by standard Linux tools and commands are documented in commercially available Linux reference manuals and on various Web sites. For more information about obtaining documentation for standard Linux administrative tasks and associated topics, see the list of Web sites and additional publications provided in "Related Software Products and Additional Publications" (page 15).

## Intended Audience

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 application code on an HP XC system.

This document assumes that the user understands, and has experience with, multiprocessor systems and the Message Passing Interface (MPI), and is familiar with HP XC architecture and concepts.

## New and Changed Information in This Edition

- The `xcxclus` and `xcxperf` utilities are now described as part of a separate document.
- There are various changes with regard to Platform LSF brand.

## Typographic Conventions

This document uses the following typographical conventions:

| | |
|---|---|
| `%`, `$`, or `#` | A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Korn, POSIX, and Bourne shells. A number sign represents the superuser prompt. |
| *audit*(5) | A manpage. The manpage name is *audit*, and it is located in Section 5. |
| `Command` | A command name or qualified command phrase. |
| `Computer output` | Text displayed by the computer. |

| | |
|---|---|
| **Ctrl+x** | A key sequence. A sequence such as **Ctrl+x** indicates that you must hold down the key labeled **Ctrl** while you press another key or mouse button. |
| `ENVIRONMENT VARIABLE` | The name of an environment variable, for example, `PATH`. |
| [ERROR NAME] | The name of an error, usually returned in the `errno` variable. |
| **Key** | The name of a keyboard key. **Return** and **Enter** both refer to the same key. |
| **Term** | The defined use of an important word or phrase. |
| **`User input`** | Commands and other text that you type. |
| *`Variable`* | The name of a placeholder in a command, function, or other syntax display that you replace with an actual value. |
| [ ] | The contents are optional in syntax. If the contents are a list separated by \|, you can choose one of the items. |
| { } | The contents are required in syntax. If the contents are a list separated by \|, you must choose one of the items. |
| . . . | The preceding element can be repeated an arbitrary number of times. |
| \| | Separates items in a list of choices. |
| WARNING | A warning calls attention to important information that if not understood or followed will result in personal injury or nonrecoverable system problems. |
| CAUTION | A caution calls attention to important information that if not understood or followed will result in data loss, data corruption, or damage to hardware or software. |
| IMPORTANT | This alert provides essential information to explain a concept or to complete a task |
| NOTE | A note contains additional information to emphasize or supplement important points of the main text. |

## HP XC and Related HP Products Information

The HP XC System Software Documentation Set, the Master Firmware List, and HP XC HowTo documents are available at this HP Technical Documentation Web site:

http://www.docs.hp.com/en/linuxhpc.html

The HP XC System Software Documentation Set includes the following core documents:

| | |
|---|---|
| *HP XC System Software Release Notes* | Describes important, last-minute information about firmware, software, or hardware that might affect the system. This document is not shipped on the HP XC documentation CD. It is available only on line. |
| *HP XC Hardware Preparation Guide* | Describes hardware preparation tasks specific to HP XC that are required to prepare each supported hardware model for installation and configuration, including required node and switch connections. |
| *HP XC System Software Installation Guide* | Provides step-by-step instructions for installing the HP XC System Software on the head node and configuring the system. |
| *HP XC System Software Administration Guide* | Provides an overview of the HP XC system administrative environment, cluster administration tasks, node maintenance tasks, LSF® administration tasks, and troubleshooting procedures. |

| | |
|---|---|
| *HP XC System Software User's Guide* | Provides an overview of managing the HP XC user environment with modules, managing jobs with LSF, and describes how to build, run, debug, and troubleshoot serial and parallel applications on an HP XC system. |
| QuickSpecs for HP XC System Software | Provides a product overview, hardware requirements, software requirements, software licensing information, ordering information, and information about commercially available software that has been qualified to interoperate with the HP XC System Software. The QuickSpecs are located on line: http://www.hp.com/go/clusters |

See the following sources for information about related HP products.

### HP XC Program Development Environment

The Program Development Environment home page provide pointers to tools that have been tested in the HP XC program development environment (for example, TotalView® and other debuggers, compilers, and so on).

http://h20311.www2.hp.com/HPC/cache/276321-0-0-0-121.html

### HP Message Passing Interface

HP Message Passing Interface (HP-MPI) is an implementation of the MPI standard that has been integrated in HP XC systems. The home page and documentation is located at the following Web site:

http://www.hp.com/go/mpi

### HP Serviceguard

HP Serviceguard is a service availability tool supported on an HP XC system. HP Serviceguard enables some system services to continue if a hardware or software failure occurs. The HP Serviceguard documentation is available at the following Web site:

http://www.docs.hp.com/en/ha.html

### HP Scalable Visualization Array

The HP Scalable Visualization Array (SVA) is a scalable visualization solution that is integrated with the HP XC System Software. The SVA documentation is available at the following Web site:

http://www.docs.hp.com/en/linuxhpc.html

### HP Cluster Platform

The cluster platform documentation describes site requirements, shows you how to set up the servers and additional devices, and provides procedures to operate and manage the hardware. These documents are available at the following Web site:

http://www.docs.hp.com/en/linuxhpc.html

### HP Integrity and HP ProLiant Servers

Documentation for HP Integrity and HP ProLiant servers is available at the following Web site:

http://www.docs.hp.com/en/hw.html

## Related Information

This section provides useful links to third-party, open source, and other related software products.

**Supplementary Software Products** This section provides links to third-party and open source software products that are integrated into the HP XC System Software core technology. In the HP XC documentation, except where necessary, references to third-party and open source software components are generic, and the HP XC adjective is not added to any reference to a third-party or open source command or product name. For example, the SLURM `srun` command is simply referred to as the `srun` command.

The location of each Web site or link to a particular topic listed in this section is subject to change without notice by the site provider.

- http://www.platform.com

  Home page for Platform Computing, Inc., the developer of the Load Sharing Facility (LSF). LSF with SLURM, the batch system resource manager used on an HP XC system, is tightly integrated with the HP XC and SLURM software. Documentation specific to LSF with SLURM is provided in the HP XC documentation set.

  Standard LSF is also available as an alternative resource management system (instead of LSF with SLURM) for HP XC. This is the version of LSF that is widely discussed on the Platform Web site.

  For your convenience, the following Platform LSF documents are shipped on the HP XC documentation CD in PDF format:

  — *Administering Platform LSF*
  — *Administration Primer*
  — *Platform LSF Reference*
  — *Quick Reference Card*
  — *Running Jobs with Platform LSF*

  LSF procedures and information supplied in the HP XC documentation, particularly the documentation relating to the LSF integration with SLURM, supersedes the information supplied in the LSF manuals from Platform Computing, Inc.

  The Platform LSF manpages are installed by default. *lsf_diff*(7) supplied by HP describes LSF command differences when using LSF with SLURM on an HP XC system.

  The following documents in the HP XC System Software Documentation Set provide information about administering and using LSF on an HP XC system:

  — *HP XC System Software Administration Guide*
  — *HP XC System Software User's Guide*

- https://computing.llnl.gov/linux/slurm/documentation.html

  Documentation for the Simple Linux Utility for Resource Management (SLURM), which is integrated with LSF to manage job and compute resources on an HP XC system.

- http://www.nagios.org/

  Home page for Nagios®, a system and network monitoring application that is integrated into an HP XC system to provide monitoring capabilities. Nagios watches specified hosts and services and issues alerts when problems occur and when problems are resolved.

- http://oss.oetiker.ch/rrdtool

  Home page of RRDtool, a round-robin database tool and graphing system. In the HP XC system, RRDtool is used with Nagios to provide a graphical view of system status.

- http://supermon.sourceforge.net/

  Home page for Supermon, a high-speed cluster monitoring system that emphasizes low perturbation, high sampling rates, and an extensible data protocol and programming interface. Supermon works in conjunction with Nagios to provide HP XC system monitoring.

- http://www.llnl.gov/linux/pdsh/

  Home page for the parallel distributed shell (`pdsh`), which executes commands across HP XC client nodes in parallel.

- http://www.balabit.com/products/syslog_ng/

  Home page for syslog-ng, a logging tool that replaces the traditional syslog functionality. The syslog-ng tool is a flexible and scalable audit trail processing tool. It provides a centralized, securely stored log of all devices on the network.
- http://systemimager.org

  Home page for SystemImager®, which is the underlying technology that distributes the golden image to all nodes and distributes configuration changes throughout the system.
- http://linuxvirtualserver.org

  Home page for the Linux Virtual Server (LVS), the load balancer running on the Linux operating system that distributes login requests on the HP XC system.
- http://www.macrovision.com

  Home page for Macrovision®, developer of the FLEXlm™ license management utility, which is used for HP XC license management.
- http://sourceforge.net/projects/modules/

  Web site for Modules, which provide for easy dynamic modification of a user's environment through modulefiles, which typically instruct the module command to alter or set shell environment variables.
- http://dev.mysql.com/

  Home page for MySQL AB, developer of the MySQL database. This Web site contains a link to the MySQL documentation, particularly the *MySQL Reference Manual*.

**Related Software Products and Additional Publications** This section provides pointers to Web sites for related software products and provides references to useful third-party publications. The location of each Web site or link to a particular topic is subject to change without notice by the site provider.

#### Linux Web Sites

- http://www.redhat.com

  Home page for Red Hat®, distributors of Red Hat Enterprise Linux Advanced Server, a Linux distribution with which the HP XC operating environment is compatible.
- http://www.linux.org/docs/index.html

  This Web site for the Linux Documentation Project (LDP) contains guides that describe aspects of working with Linux, from creating your own Linux system from scratch to bash script writing. This site also includes links to Linux HowTo documents, frequently asked questions (FAQs), and manpages.
- http://www.linuxheadquarters.com

  Web site providing documents and tutorials for the Linux user. Documents contain instructions on installing and using applications for Linux, configuring hardware, and a variety of other topics.
- http://www.gnu.org

  Home page for the GNU Project. This site provides online software and information for many programs and utilities that are commonly used on GNU/Linux systems. Online information include guides for using the bash shell, emacs, make, cc, gdb, and more.

### MPI Web Sites

- http://www.mpi-forum.org

  Contains the official MPI standards documents, errata, and archives of the MPI Forum. The MPI Forum is an open group with representatives from many organizations that define and maintain the MPI standard.
- http://www-unix.mcs.anl.gov/mpi/

  A comprehensive site containing general information, such as the specification and FAQs, and pointers to other resources, including tutorials, implementations, and other MPI-related sites.

### Compiler Web Sites

- http://www.intel.com/software/products/compilers/index.htm

  Web site for Intel® compilers.
- http://support.intel.com/support/performancetools/

  Web site for general Intel software development information.
- http://www.pgroup.com/

  Home page for The Portland Group™, supplier of the PGI® compiler.

### Debugger Web Site

http://www.etnus.com

Home page for Etnus, Inc., maker of the TotalView® parallel debugger.

### Software RAID Web Sites

- http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html and http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/Software-RAID-HOWTO.pdf

  A document (in two formats: HTML and PDF) that describes how to use software RAID under a Linux operating system.
- http://www.linuxdevcenter.com/pub/a/linux/2002/12/05/RAID.html

  Provides information about how to use the `mdadm` RAID management utility.

### Additional Publications

For more information about standard Linux system administration or other related software topics, consider using one of the following publications, which must be purchased separately:

- *Linux Administration Unleashed*, by Thomas Schenk, et al.
- *Linux Administration Handbook*, by Evi Nemeth, Garth Snyder, Trent R. Hein, et al.
- *Managing NFS and NIS*, by Hal Stern, Mike Eisler, and Ricardo Labiaga (O'Reilly)
- *MySQL*, by Paul Debois
- *MySQL Cookbook*, by Paul Debois
- *High Performance MySQL*, by Jeremy Zawodny and Derek J. Balling (O'Reilly)
- *Perl Cookbook, Second Edition*, by Tom Christiansen and Nathan Torkington
- *Perl in A Nutshell: A Desktop Quick Reference* , by Ellen Siever, et al.

## Manpages

Manpages provide online reference and command information from the command line. Manpages are supplied with the HP XC system for standard HP XC components, Linux user commands, LSF commands, and other software components that are distributed with the HP XC system.

Manpages for third-party software components might be provided as a part of the deliverables for that component.

Using *discover*(8) as an example, you can use either one of the following commands to display a manpage:

```
$ man discover
$ man 8 discover
```

If you are not sure about a command you need to use, enter the `man` command with the `-k` option to obtain a list of commands that are related to a keyword. For example:

```
$ man -k keyword
```

## HP Encourages Your Comments

HP encourages comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to:

**docsfeedback@hp.com**

Include the document title, manufacturing part number, and any comment, error found, or suggestion for improvement you have concerning this document.

# 1 Overview of the User Environment

The HP XC system is a collection of computer nodes, networks, storage, and software, built into a **cluster**, that work together. It is designed to maximize workload and I/O performance, and to provide the efficient management of large, complex, and dynamic workloads.

This chapter addresses the following topics:

## 1.1 System Architecture

The HP XC architecture is designed as a clustered system with single system traits. From a user perspective, this architecture achieves a single system view, providing capabilities such as the following:

- Single user login
- Single file system namespace
- Integrated view of system resources
- Integrated program development environment
- Integrated job submission environment

### 1.1.1 HP XC System Software

The HP XC System Software enables the nodes in the platform to run cohesively to achieve a single system view. You can determine the version of the HP XC System Software from the `/etc/hptc-release` file.

```
$ cat /etc/hptc-release
HP XC V#.# RCx PKn date
```

Where:

| | |
|---|---|
| `#.#` | Is the version of the HP XC System Software |
| `RCx` | Is the release candidate of the version. |
| `PKnn` | Indicates there was a cumulative patch kit for the HP XC System Software installed.<br>For example, if `PK02` appears in the output, it indicates that both cumulative patch kits PK01 and PK02 have been installed.<br>This field is blank if no patch kits are installed. |
| `date` | Is the date (in *yyyymmdd* format) the software was released. |

### 1.1.2 Operating System

The HP XC system is a high-performance compute cluster that runs HP XC Linux for High Performance Computing Version 1.0 (HPC Linux) as its software base. Any serial or thread-parallel applications, or applications built shared with HP-MPI that run correctly on Red Hat Enterprise Linux Advanced Server Version 3.0 or Version 4.0, also run correctly on HPC Linux.

### 1.1.3 Node Platforms

The HP XC System Software is available on several platforms. You can determine the platform by examining the top few fields of the `/proc/cpuinfo` file, for example, by using the `head` command:

```
$ head /proc/cpuinfo
```

Table 1-1 presents the representative output for each of the platforms. This output may differ according to changes in models and so on.

**Table 1-1 Determining the Node Platform**

| Platform | Partial Output of `/proc/cpuinfo` |
|---|---|
| CP3000 | `processor : 0`<br>`vendor_id : GenuineIntel`<br>`cpu family : 15`<br>`model : 3`<br>`model name : Intel(R) Xeon(TM)` |
| CP4000 | `processor : 0`<br>`vendor_id : AuthenticAMD`<br>`cpu family : 15`<br>`model : 5`<br>`model name : AMD Opteron(tm)` |
| CP6000 | `processor : 0`<br>`vendor : GenuineIntel`<br>`arch : IA-64`<br>`family : Itanium 2`<br>`model : 1` |
| CP300BL (Blade-only XC systems) | `processor : 0`<br>`vendor_id : GenuineIntel`<br>`cpu family : 15`<br>`model : 6`<br>`model name : Intel(R) Xeon(TM) CPU 3.73GHz` |

**Note:**

The `/proc/cpuinfo` file is dynamic.

## 1.1.4 Node Specialization

The HP XC system is implemented as a *sea-of-nodes*. Each node in the system contains the same software image on its local disk. There are two types of nodes in the system — a **head node** and **client nodes**.

| | |
|---|---|
| head node | The node is installed with the HP XC system software first — it is used to generate other HP XC (client) nodes. The head node is generally of interest only to the administrator of the HP XC system. |
| client nodes | All the other the nodes that make up the system. They are replicated from the head node and are usually given one or more specialized **roles** to perform various system functions, such as logging into the system or running jobs. |

The HP XC system allows for the specialization of client nodes to enable efficient and flexible distribution of the workload. Nodes can be assigned one or more specialized **roles** that determine how a particular node is used and what system services it provides. Of the many different roles that can be assigned to a client node, the following roles contain services that are of special interest to the general user:

**login role** The role most visible to users is on nodes that have the login role. Nodes with the login role are where you log in and interact with the system to perform various tasks. For example, once logged in to a node with login role, you can execute commands, build applications, or submit jobs to compute nodes for execution. There can be one or several nodes with the login role in an HP XC system, depending upon cluster size and requirements. Nodes with the login role are a part of the Linux Virtual Server ring, which

| | |
|---|---|
| | distributes login requests from users. A node with the login role is referred to as a **login node** in this manual. |
| **compute role** | The compute role is assigned to nodes where jobs are to be distributed and run. Although all nodes in the HP XC system are capable of carrying out computations, the nodes with the compute role are the primary nodes used to run jobs. Nodes with the compute role become a part of the resource pool used by LSF and SLURM, which manage and distribute the job workload. Jobs that are submitted to compute nodes must be launched from nodes with the login role. Nodes with the compute role are referred to as **compute nodes** in this manual. |

## 1.1.5 Storage and I/O

The HP XC system supports both shared (global) and private (local) disks and file systems.

Shared file systems can be mounted on all the other nodes by means of Lustre™ or NFS. This gives users a single view of all the shared data on disks attached to the HP XC system.

### SAN Storage

The HP XC system uses the HP StorageWorks Scalable File Share (HP StorageWorks SFS), which is based on Lustre technology and uses the Lustre File System from Cluster File Systems, Inc. This is a turnkey Lustre system from HP. It supplies access to Lustre file systems through Lustre client/server protocols over various system interconnects. The HP XC system is a client to the HP StorageWorks SFS server.

### Local Storage

Local storage for each node holds the operating system, a copy of the HP XC System Software, and temporary space that can be used by jobs running on the node.

HP XC file systems are described in detail in "File System".

## 1.1.6 File System

Each node of the HP XC system has its own local copy of all the HP XC System Software files including the Linux distribution; it also has its own local user files. Every node can also import files from NFS or Lustre file servers. HP XC System Software uses NFS 3, including both client and server functionality. HP XC System Software also enables Lustre client services for high-performance and high-availability file I/O. These Lustre client services require the separate installation of Lustre software, provided with the HP Storage Works Scalable File Share (SFS).

NFS files can be shared exclusively among the nodes of the HP XC System or can be shared between the HP XC and external systems. External NFS files can be shared with any node having a direct external network connection. It is also possible to set up NFS to import external files to HP XC nodes without external network connections, by routing through a node with an external network connection. Your system administrator can choose to use either the HP XC administrative network or the HP XC system **interconnect** for NFS operations. The HP XC system interconnect can potentially offer higher performance, but only at the potential decrease in the performance of application communications.

For high-performance or high-availability file I/O, the Lustre file system is available on HP XC. The Lustre file system uses POSIX-compliant syntax and semantics. The HP XC System Software includes kernel modifications required for Lustre client services which enables the operation of the separately installable Lustre client software. The Lustre file server product used on HP XC is the HP StorageWorks Scalable File Share (SFS), which fully supports the HP XC System Software.

The SFS also includes HP XC Lustre client software. The SFS can be integrated with the HP XC so that Lustre I/O is performed over the same high-speed system **interconnect** fabric used by the HP XC. So, for example, if the HP XC system interconnect is based on a Quadrics® QsNet II® switch, then the SFS will serve files over ports on that switch. The file operations are able to proceed at the full bandwidth of the HP XC system interconnect because these operations are implemented directly over the low-level communications libraries. Further optimizations of file I/O can be achieved at the application level using special file system commands – implemented as `ioctls` – which allow a program to interrogate the attributes of the file system, modify the stripe size and other attributes of new (zero-length) files, and so on. Some of these optimizations are implicit in the HP-MPI I/O library, which implements the MPI-2 file I/O standard.

#### File System Layout

In an HP XC system, the basic file system layout is the same as that of the Red Hat Advanced Server 3.0 Linux file system.

The HP XC file system is structured to separate cluster-specific files, base operating system files, and user-installed software files. This allows for flexibility and ease of potential upgrades of the system software and keeps software from conflicting with user installed software. Files are segregated into the following types and locations:

- Software specific to HP XC is located in `/opt/hptc`
- HP XC configuration data is located in `/opt/hptc/etc`
- Clusterwide directory structure (file system) is located in `/hptc_cluster`

Be aware of the following information about the HP XC file system layout:

- Open source software that by default would be installed under the `/usr/local` directory is instead installed in the `/opt/hptc` directory.
- Software installed in the `/opt/hptc` directory is not intended to be updated by users.
- Software packages are installed in directories under the `/opt/hptc` directory under their own names. The exception to this is third-party software, which usually goes in `/opt/r`.
- Four directories under the `/opt/hptc` directory contain symbolic links to files included in the packages:
  - `/opt/hptc/bin`
  - `/opt/hptc/sbin`
  - `/opt/hptc/lib`
  - `/opt/hptc/man`

  Each package directory should have a directory corresponding to each of these directories in which every file has a symbolic link created in the `/opt/hptc/` directory.

## 1.1.7 System Interconnect Network

The HP XC system interconnect provides high-speed connectivity for parallel applications. The system **interconnect network** provides a high-speed communications path used primarily for user file service and for communications within user applications that are distributed among nodes of the system. The system interconnect network is a private network within the HP XC. Typically, every node in the HP XC is connected to the system interconnect.

Table 1-2 indicates the types of system interconnects that are used on HP XC systems.

**Table 1-2 HP XC System Interconnects**

| | CP3000 | CP4000 | CP6000 |
|---|---|---|---|
| Quadrics QSNet II® | | X | X |
| Myrinet® | X | X | |
| Gigabit Ethernet® | X | X | X |
| InfiniBand® | X | X | X |

Additional information on supported system interconnects is provided in the *HP XC Hardware Preparation Guide*.

### 1.1.8 Network Address Translation (NAT)

The HP XC system uses Network Address Translation (**NAT**) to enable nodes in the HP XC system that do not have direct external network connections to open outbound network connections to external network resources.

## 1.2 Determining System Configuration Information

You can determine various system configuration parameters with a few commands:

| | |
|---|---|
| Use the following command to display the **version** of the HP XC System Software: | `cat /etc/hptc-release` |
| Use either of these commands to display the **Kernel version**: | `uname -rcat /proc/version` |
| Use the following command to display the **RPM**s: | `rpm -qa` |
| Use the following command to display the amount of free and used **memory** in megabytes: | `free -m` |
| Use the following command to display the **disk partitions** and their sizes: | `cat /proc/partitions` |
| Use the following command to display the **swap** usage summary by device: | `swapon -s` |
| Use the following commands to display the **cache** information; this is not available on all systems. | `cat /proc/pal/cpu0/cache_info` `cat /proc/pal/cpu1/cache_info` |

## 1.3 User Environment

This section introduces some general information about logging in, configuring, and using the HP XC environment.

### 1.3.1 LVS

The HP XC system uses the Linux Virtual Server (**LVS**) to present a single **host name** for user logins. LVS is a highly scalable virtual server built on a system of real servers. By using LVS, the architecture of the HP XC system is transparent to end users, and they see only a single virtual server. This eliminates the need for users to know how the system is configured in order to successfully log in and use the system. Any changes in the system configuration are transparent to end users. LVS also provides load balancing across login nodes, which distributes login requests to different servers.

### 1.3.2 Modules

The HP XC system provides the Modules Package (not to be confused with Linux kernel modules) to configure and modify the user environment. The Modules Package enables dynamic modification of a user's environment by means of modulefiles. Modulefiles provide a convenient means for users to tailor their working environment as necessary. One of the key features of modules is to allow multiple versions of the same software to be used in a controlled manner.

A **modulefile** contains information to configure the shell for an application. Typically, a modulefile contains instructions that alter or set shell environment variables, such as `PATH` and `MANPATH`, to enable access to various installed software. Many users on a system can share modulefiles, and users may have their own collection to supplement or replace the shared modulefiles.

Modulefiles can be loaded into the your environment automatically when you log in to the system, or any time you need to alter the environment. The HP XC system does not preload modulefiles.

See Chapter 3 “Configuring Your Environment with Modulefiles” for more information.

### 1.3.3 Commands

The HP XC user environment includes standard Linux commands, LSF commands, SLURM commands, HP-MPI commands, and modules commands. This section provides a brief overview of these command sets.

| | |
|---|---|
| Linux commands | You can use standard Linux user commands and tools on the HP XC system. Standard Linux commands are not described in this document, but you can access Linux command descriptions in Linux documentation and manpages. Run the Linux `man` command with the Linux command name to display the corresponding manpage. |
| LSF commands | HP XC supports LSF and the use of **standard LSF** commands, some of which operate differently in the HP XC environment from standard LSF behavior. The use of LSF commands in the HP XC environment is described in Chapter 10 “Using LSF”, and in the HP XC `lsf_diff` manpage. Information about standard LSF commands is available in Platform LSF documentation, and in the LSF manpages. For your convenience, the HP XC Documentation CD contains LSF manuals from Platform Computing, Inc. LSF manpages are available on the HP XC system. |
| SLURM commands | HP XC uses the Simple Linux Utility for Resource Management (SLURM) for system resource management and job scheduling. Standard SLURM commands are available through the command line. SLURM functionality is described in Chapter 9 “Using SLURM”. Descriptions of SLURM commands are available in the SLURM manpages. Invoke the `man` command with the SLURM command name to access them. |
| HP-MPI commands | You can run standard HP-**MPI** commands from the command line. Descriptions of HP-MPI commands are available in the HP-MPI documentation, which is supplied with the HP XC system software. |
| Modules commands | The HP XC system uses standard Modules commands to load and unload modulefiles, which are used to configure and modify the user environment. Modules commands are described in “Overview of Modules”. |

## 1.4 Application Development Environment

The HP XC system provides an environment that enables developing, building, and running applications using multiple nodes with multiple cores. These applications can range from parallel applications using many cores to **serial application**s using a single core.

### 1.4.1 Parallel Applications

The HP XC **parallel application** development environment allows parallel application processes to be started and stopped together on a large number of application processors, along with the I/O and process control structures to manage these kinds of applications.

Full details and examples of how to build, run, debug, and troubleshoot parallel applications are provided in “Developing Parallel Applications”.

### 1.4.2 Serial Applications

You can build and run **serial application**s under the HP XC development environment. A serial application is a command or application that does not use any form of parallelism.

Full details and examples of how to build, run, debug, and troubleshoot serial applications are provided in "Building Serial Applications".

## 1.5 Run-Time Environment

This section describes LSF, SLURM, and HP-**MPI**, and how these components work together to provide the HP XC run-time environment. LSF focuses on scheduling (and managing the workload) and SLURM provides efficient and scalable resource management of the compute nodes.

Another HP XC environment features **standard LSF** without the interaction with the SLURM resource manager.

### 1.5.1 SLURM

Simple Linux Utility for Resource Management (SLURM) is a resource management system that is integrated into the HP XC system. SLURM is suitable for use on large and small Linux clusters. It was developed by Lawrence Livermore National Lab and Linux Networks. As a resource manager, SLURM allocates exclusive or unrestricted access to resources (application and compute nodes) for users to perform work, and provides a framework to start, execute and monitor work (normally a parallel job) on the set of allocated nodes.

A SLURM system consists of two daemons, one configuration file, and a set of commands and APIs. The central controller daemon, `slurmctld`, maintains the global state and directs operations. A `slurmd` daemon is deployed to each computing node and responds to job-related requests, such as launching jobs, signalling, and terminating jobs. End users and system software (such as LSF) communicate with SLURM by means of commands or APIs — for example, allocating resources, launching parallel jobs on allocated resources, and terminating running jobs.

SLURM groups compute nodes (the nodes where jobs are run) together into "partitions". The HP XC system can have one or several partitions. When HP XC is installed, a single partition of compute nodes is created by default for LSF batch jobs. The system administrator has the option of creating additional partitions. For example, another partition could be created for interactive jobs.

### 1.5.2 Load Sharing Facility (LSF)

The Load Sharing Facility (LSF) from Platform Computing, Inc. is a batch system resource manager that has been integrated with SLURM for use on the HP XC system. LSF for SLURM is included with the HP XC System Software, and is an integral part of the HP XC environment. LSF interacts with SLURM to obtain and allocate available resources, and to launch and control all the jobs submitted to LSF. LSF accepts, queues, schedules, dispatches, and controls all the batch jobs that users submit, according to policies and configurations established by the HP XC site administrator. On an HP XC system, LSF for SLURM is installed and runs on one HP XC node, known as the **LSF execution host**.

A complete description of LSF is provided in Chapter 10 "Using LSF". In addition, for your convenience, the HP XC Documentation CD contains LSF manuals from Platform Computing.

### 1.5.3 Standard LSF

Standard LSF is also available on the HP XC system. The information for using **standard LSF** is documented in the LSF manuals from Platform Computing. For your convenience, the HP XC documentation CD contains these manuals.

## 1.5.4 How LSF and SLURM Interact

In the HP XC environment, LSF cooperates with SLURM to combine the powerful scheduling functionality of LSF with the scalable parallel job launching capabilities of SLURM. LSF acts primarily as a workload scheduler on top of the SLURM system, providing policy and topology-based scheduling for end users. SLURM provides an execution and monitoring layer for LSF. LSF uses SLURM to detect system topology information, make scheduling decisions, and launch jobs on allocated resources.

When a job is submitted to LSF, LSF schedules the job based on job resource requirements. LSF communicates with SLURM to allocate the required HP XC compute nodes for the job from the SLURM `lsf` partition. LSF provides node-level scheduling for parallel jobs, and core-level scheduling for serial jobs. Because of node-level scheduling, a parallel job may be allocated more cores than it requested, depending on its resource request; the `srun` or `mpirun -srun` launch commands within the job still honor the original request. LSF always tries to pack multiple serial jobs on the same node, with one core per job. Parallel jobs and serial jobs cannot coexist on the same node.

After the LSF scheduler allocates the SLURM resources for a job, the SLURM allocation information is recorded with the job. You can view this information with the `bjobs` and `bhist` commands.

When LSF starts a job, it sets the `SLURM_JOBID` and `SLURM_NPROCS` environment variables in the job environment. `SLURM_JOBID` associates the LSF job with SLURM's allocated resources. The `SLURM_NPROCS` environment variable is set to the originally requested number of cores. LSF dispatches the job from the **LSF execution host**, which is the same node on which LSF daemons run. The LSF `JOB_STARTER` script, which is configured for all queues, uses the `srun` command to launch a user job on the first node in the allocation. Your job can contain additional `srun` or `mpirun` commands to launch tasks to all nodes in the allocation.

While a job is running, all resource limits supported by LSF enforced, including core limit, CPU time limit, data limit, file size limit, memory limit and stack limit. When you terminate a job, LSF uses the SLURM `scancel` command to propagate the signal to the entire job.

After a job finishes, LSF releases all allocated resources.

A detailed description, along with an example and illustration, of how LSF and SLURM cooperate to launch and manage jobs is provided in "How LSF and SLURM Launch and Manage a Job". It is highly recommended that you review this information.

In summary, and in general:

| | |
|---|---|
| LSF | Determines WHEN and WHERE the job will run. LSF communicates with SLURM to determine WHICH resources are available, and SELECTS the appropriate set of nodes for the job. |
| SLURM | Allocates nodes for jobs as determined by LSF. It CONTROLS task/rank distribution within the allocated nodes. SLURM also starts the executables on each host as requested by the HP-MPI `mpirun` command. |
| HP-MPI | Determines HOW the job runs. It is part of the application, so it performs communication. HP-**MPI** can also pinpoint the processor on which each rank runs. |

## 1.5.5 HP-MPI

HP-**MPI** is a high-performance implementation of the Message Passing Interface (MPI) standard and is included with the HP XC system. HP-MPI uses SLURM to launch jobs on an HP XC system — however, it manages the global MPI exchange so that all processes can communicate with each other.

See the HP-MPI documentation for more information.

## 1.6 Components, Tools, Compilers, Libraries, and Debuggers

This section provides a brief overview of the some of the common tools, compilers, libraries, and debuggers available for use on HP XC.

An HP XC system is integrated with several open source software components. HP XC incorporates a Linux operating system and its standard commands and tools, and maintains the Linux ABI. In addition, HP XC incorporates LSF and SLURM to launch and manage jobs, and includes HP-**MPI** for high-performance, parallel, message-passing applications.

You can use most standard open source compilers and tools on an HP XC system; however, you must purchase them separately. Several open source and commercially available software packages have been tested with the HP XC Software. The following lists some of the software packages that have been tested for use with HP XC. This list provides a sample of what is available on HP XC and is not intended as a complete list. Some packages listed are actually included as part of the HPC Linux distribution and as such can be used as part of the HP XC development environment. The tested software packages include, but are not limited to, the following:

- Intel Fortran 95, C, C++ Compiler Version 7.1, 8.0, 9.0, 9.1 including OpenMP, for Itanium® (includes ldb debugger)
- gcc version 3.2.3 (included in the HP XC distribution)
- g77 version 3.2.3 (included in the HP XC distribution)
- Portland Group PGI Fortran90, C, C++ Version 5.1, including OpenMP, for CP4000
- Quadrics SHMEM, as part of QsNet II user libraries, on Itanium systems connected with the Quadrics QsNet II switch (included in the HP XC distribution)
- Etnus TotalView debugger Version 6.4
- gdb (part of the HP XC Linux distribution)
- Intel MKL V6.1 on Itanium
- AMD Math Core Library Version 2.0 on CP4000
- valgrind 2.0.0 (http://valgrind.kde.org) in 32-bit mode only
- oprofile 0.7.1 (http://oprofile.sourceforge.net)
- PAPI 3.2 (http://icl.cs.utk.edu/papi)
- Intel Visual Analyzer/Tracer (formally Pallas Vampir and Vampirtrace performance analyzer ) on Itanium
- GNU make, including distributed parallel make (included in the HP XC distribution)

Other standard tools and libraries are available, and you can most likely used them on HP XC as you would on any other standard Linux system.

**IMPORTANT:** Software that is not described in the HP XC QuickSpecs or in the HP XC documentation set has not been tested with the HP XC System Software. HP does not warrant the operation of any software that is not specifically identified in the product specification (the HP XC QuickSpecs) and is not documented in the HP XC documentation set.

# 2 Using the System

This chapter describes the tasks and commands that the general user must know to use the system. It addresses the following topics:

- "Logging In to the System" (page 29)
- "Overview of Launching and Managing Jobs" (page 29)
- "Performing Other Common User Tasks" (page 31)
- "Getting System Help and Information" (page 32)

## 2.1 Logging In to the System

Logging in to an HP XC system is similar to logging in to any standard Linux system. You can only login on nodes that have the login **role**; those nodes are enabled for logins.

### 2.1.1 LVS Login Routing

The HP XC system uses the Linux Virtual Server (**LVS**) facility to present a set of login nodes with a single **cluster** name. When you log in to the system, LVS automatically routes your login request to an available login node on the system. LVS load balances login sessions across the login nodes and improves the availability of login access. When you log in to the HP XC system, you do not have to know specific node names to log in, only the HP XC system's cluster name.

### 2.1.2 Using the Secure Shell to Log In

Secure Shell (**ssh**) is the preferred method for accessing the HP XC system.

Typically, you access the HP XC system using the `ssh` command to get a login shell or to execute commands. For example:

```
$ ssh user-name@system-name
user-name@system-name's password:
```

The **ssh** service also allows file transfer using the `scp` or `sftp` commands over the same port as `ssh`.

The typical `r*` UNIX commands, such as `rlogin`, `rsh`, and `rcp`, are not installed on an HP XC system by default because of their inherent insecurity. The `ssh` command transfers all login and password information in an encrypted form instead of the plaintext form used by the `r*` UNIX commands (as well as `telnet` and `ftp`).

If you want to use `ssh` without password prompting, you must set up `ssh` authentication keys. See *ssh*(1) for information about using `ssh` authentication keys.

The Secure Shell is further discussed in "Enabling Remote Execution with OpenSSH" (page 107).

You can bypass the need to enter a login and password each time you log in by updating the `ssh` keys. The `ssh_create_shared_keys` command, used on a one-time basis, updates the appropriate `ssh` key files in your `$HOME/.ssh` directory, provided:

- You have a user account on the HP XC system.
- Your home directory is mounted on all the nodes to which you log in or on which you launch jobs.

Ensure that you deny write permission on your home directory to the group and others; otherwise the `ssh` command . The following command accomplishes that for you:

```
$ chmod go -w $HOME
```

## 2.2 Overview of Launching and Managing Jobs

This section provides a brief description of some of the many ways to launch jobs, manage jobs, and get information about jobs on an HP XC system. This section is intended only as a quick overview about some basic ways of running and managing jobs. Full information and details about the HP XC job launch environment are provided in "Using SLURM") and the LSF section of "Using LSF") of this document.

## 2.2.1 Introduction

As described in "Run-Time Environment" (page 25), SLURM and LSF cooperate to run and manage jobs on the HP XC system, combining LSF's powerful and flexible scheduling functionality with SLURM's scalable parallel job-launching capabilities.

SLURM is the low-level resource manager and job launcher, and performs core allocation for jobs. LSF gathers information about the cluster from SLURM. When a job is ready to be launched, LSF creates a SLURM node allocation and dispatches the job to that allocation.

Although you can launch jobs directly using SLURM, HP recommends that you use LSF to take advantage of its scheduling and job management capabilities. You can add SLURM options to the LSF job launch command line to further define job launch requirements. Use the HP-MPI `mpirun` command and its options within LSF to launch jobs that require MPI's high-performance message-passing capabilities.

When the HP XC system is installed, a SLURM partition of nodes is created to contain LSF jobs. This partition is called the `lsf` partition.

When a job is submitted to LSF, the LSF scheduler prioritizes the job and waits until the required resources (compute nodes from the `lsf` partition) are available.

When the requested resources are available for the job, LSF creates a SLURM allocation of nodes on behalf of the user, sets the SLURM `JobID` for the allocation, and dispatches the job with the LSF `JOB_STARTER` script to the first allocated node.

A detailed explanation of how SLURM and LSF interact to launch and manage jobs is provided in "How LSF and SLURM Launch and Manage a Job" (page 92).

## 2.2.2 Getting Information About Queues

The LSF `bqueues` command lists the configured job queues in LSF. By default, `bqueues` returns the following information about all queues:

- Queue name
- Queue priority
- Queue status
- Job slot statistics
- Job state statistics

To get information about queues, enter the `bqueues` as follows:

```
$ bqueues
```

For more information about using this command and a sample of its output, see "Examining System Queues" (page 96)

## 2.2.3 Getting Information About Resources

The LSF `bhosts`, `lshosts`, and `lsload` commands are quick ways to get information about system resources. LSF daemons run on only one node in the HP XC system, so the `bhosts` and `lshosts` commands will list one host — which represents all the resources of the HP XC system. The total number of cores for that host should be equal to the total number of cores assigned to the SLURM `lsf` partition.

- The LSF `bhosts` command provides a summary of the jobs on the system and information about the current state of LSF.

  ```
  $ bhosts
  ```

For more information about using this command and a sample of its output, see "Examining System Core Status" (page 95)

- The LSF `lshosts` command displays machine-specific information for the LSF execution host node.

  ```
  $ lshosts
  ```

  For more information about using this command and a sample of its output, see "Getting Information About the LSF Execution Host Node" (page 95) .

- The LSF `lsload` command displays load information for the LSF execution host node.

  ```
  $ lsload
  ```

  For more information about using this command and a sample of its output, see "Getting Host Load Information" (page 96).

### 2.2.4 Getting Information About System Partitions

You can view information about system partitions with the SLURM `sinfo` command. The `sinfo` command reports the state of all partitions and nodes managed by SLURM and provides a wide variety of filtering, sorting, and formatting options. The `sinfo` command displays a summary of available partition and node (not job) information, such as partition names, nodes per partition, and cores per node).

```
$ sinfo
```

For more information about using the `sinfo` command and a sample of its output, see "Getting Information About the `lsf` Partition" (page 96) .

### 2.2.5 Launching Jobs

To launch a job on an HP XC system, use the LSF `bsub` command. The `bsub` command submits batch jobs or interactive batch jobs to an LSF queue for execution.

See "Submitting Jobs" (page 49) for full information about launching jobs.

### 2.2.6 Getting Information About Your Jobs

Use the LSF `bjobs` and `bhist` commands to obtain information about your running or completed jobs:

- Use the `bjobs` command to examine the status of a running job ("Examining the Status of a Job" (page 98)).
- Use the `bhist` command to obtain brief or full information about finished jobs ("Viewing the Historical Information for a Job" (page 99))

You can view the components of the actual SLURM allocation command with the LSF `bjobs -l` and `bhist -l` commands.

### 2.2.7 Stopping and Suspending Jobs

You can suspend or stop your jobs with the `bstop` and `bkill` commands:

- Use the `bstop` command to stop or suspend an LSF job.
- Use the `bkill` command to kill an LSF job.

### 2.2.8 Resuming Suspended Jobs

Use the LSF `bresume` command to resume a stopped or suspended job.

## 2.3 Performing Other Common User Tasks

This section contains general information about using the HP XC system.

### 2.3.1 Determining the LSF Cluster Name and the LSF Execution Host

The `lsid` command returns the LSF cluster name, the LSF version, and the name of the LSF execution host:

```
$ lsid
Platform LSF HPC version, Update n, build date stamp
Copyright 1992-2008 Platform Computing Corporation

My cluster name is hptclsf
My master name is lsfhost.localdomain
```

In this example, `hptclsf` is the LSF cluster name, and `lsfhost.localdomain` is the name of the virtual IP address used by the node where LSF is installed and running (LSF execution host).

## 2.4 Getting System Help and Information

In addition to the hardcopy documentation described in the preface of this document ("About This Document"), the HP XC system also provides system help and information in the form on online manpages.

Manpages provide online reference and command information from the system command line. Manpages are supplied with the HP XC system for standard HP XC components, Linux user commands, LSF commands, SLURM commands, and other software components that are distributed with the HP XC system, such as HP-MPI. Manpages for third-party vendor software components may be provided as a part of the deliverables for that software component.

To access manpages, type the `man` command with the name of a command. For example:

```
$ man sinfo
```

This command accesses the manpage for the SLURM `sinfo` command.

If you are unsure of the name of the command that you need to reference, you can enter the `man` command with the `-k` option and a keyword, to get a list of commands that are related to the keyword you entered. For example:

```
$ man -k keyword
```

The `MANPATH` environment variable is a list of directories that the `man` command uses to find a manpage. You may need to add to your `MANPATH` variable to identify the location of manpages for additional software packages.

# 3 Configuring Your Environment with Modulefiles

The HP XC system supports the use of Modules software to make it easier to configure and modify the your environment. Modules software enables dynamic modification of your environment by the use of modulefiles.

This chapter addresses the following topics:

## 3.1 Overview of Modules

A modulefile contains information to configure the shell for an application. Typically, a modulefile contains instructions that alter or set shell environment variables, such as `PATH` and `MANPATH`, to enable access to various installed software.

Modules enable multiple versions of the same software to be used in your environment in a controlled manner. For example, two different versions of the Intel C compiler can be installed on the system at the same time – the version used is based upon which Intel C compiler modulefile is loaded.

The HP XC software provides a number of modulefiles. You can also create your own modulefiles. Modulefiles can be shared by many users on a system, and users can have their own collections of modulefiles to supplement or replace the shared modulefiles.

For further information about the Modules software supplied with the HP XC system, see the Modules Web site at the following URL:

http://sourceforge.net/projects/modules/

A modulefile does not provide configuration of your environment until it is explicitly loaded. That is, the specific modulefile for a software product or application must be loaded in your environment (with the `module load` command) before the configuration information in the modulefile is effective.

You or your system administrator can configure your environment so that any desired modulefiles are automatically loaded for you when you log in to the system. You can also load a modulefile yourself, as described in "Loading a Modulefile" (page 36).

The Modules software is initialized when you log in to the HP XC system. It provides access to the commands that you can use to display information about modulefiles, load or unload modulefiles, or view a list of available modulefiles.

Modulefiles do not affect packages other than their intended package. For example, a modulefile for a compiler will not adjust `MPI_CC` (the environment variable used by HP-MPI to control which compiler to use). A modulefile for a compiler simply makes it easier to access that particular compiler; it does not try to determine how the compiler will be used.

Similarly, a modulefile for HP-MPI will not try to adjust `LD_LIBRARY_PATH` to correspond to the compiler that the `mpicc` command uses. The modulefile for **MPI** simply makes it easier to access the `mpi**` scripts and libraries. You can specify the compiler it uses through a variety of mechanisms long after the modulefile is loaded.

The previous scenarios were chosen in particular because the HP-MPI `mpicc` command uses heuristics to try to find a suitable compiler when MPI_CC or other default-overriding mechanisms are not in effect. It is possible that `mpicc` will choose a compiler inconsistent with the most recently loaded compiler module. This could cause inconsistencies in the use of shared objects. If you have multiple compilers (perhaps with incompatible shared objects) installed, it is probably wise to set `MPI_CC` (and others) explicitly to the commands made available by the compiler's modulefile.

The contents of the modulefiles in the `modulefiles_hptc` **RPM** use the vendor-intended location of the installed software. In many cases, this is under the `/opt` directory, but in a few cases (for example, the PGI compilers and the TotalView debugger) this is under the `/usr` directory.

If you install a software package other than in the intended place, you must create or edit an appropriate modulefile under the `/opt/modules/modulefiles` directory.

For the packages that install by default into the `/usr` directory (currently the PGI compilers and TotalView), their corresponding modulefiles will try their vendor-intended location under the `/usr` directory. If they do not find that directory, the packages will also search under the `/opt` directory. Therefore, you do not need to make any changes to the modulefiles if you want to install third-party software consistently as the vendor intended or consistently under the `/opt` directory,

If the package is the stable product intended to be used by the site, editing an existing modulefile is appropriate. While each modulefile has its unique characteristics, they all set some variables describing the top-level directory, and editing to adjust the string should be sufficient. You may need to repeat the adjustment if you update the `modulefiles_hptc` RPM or otherwise rebuild your system.

If the package is a variant, for example, a beta version of a compiler, first copy the default modulefile to a well-named copy, then edit the copy. You need root access to modify the modulefiles, which is generally needed to install packages in either the `/opt` or `/usr` directories.

If you download a package into a private directory, you can create a private modulefiles directory. You can then copy the corresponding default modulefile from under the `/opt/modules/modulefiles` directory into a private modulefiles directory, edit the file, and then register the directory with the `module use` command.

## 3.2 Supplied Modulefiles

The HP XC system provides the Modules Package (not to be confused with Linux kernel modules) to configure and modify the user environment. The Modules Package enables dynamic modification of a user's environment by means of modulefiles.

The HP XC system supplies the modulefiles listed in Table 3-1.

**Table 3-1 Supplied Modulefiles**

| Modulefile | Sets the HP XC User Environment to Use: |
|---|---|
| `gcc/3.4/default` | GCC C compiler, Version 3.4. |
| `gcc/4.0/default` | GCC compiler, Version 4.0. |
| `gcc/4.1/default` | GCC compiler, Version 4.1. |
| `hpcpi` | HP-CPI profiling tools. |
| `hptc` | HPTC loader. |
| `icc/8.0/default` | Intel C/C++ Version 8.0 compilers. |

**Table 3-1 Supplied Modulefiles** *(continued)*

| Modulefile | Sets the HP XC User Environment to Use: |
|---|---|
| `icc/8.1/default` | Intel C/C++ Version 8.1 compilers. |
| `icc/9.0/default` | Intel C/C++ Version 9.0 compilers. |
| `icc/9.1/default` | Intel C/C++ Version 9.1 compilers. |
| `idb/7.3/default` | Intel IDB debugger. |
| `idb/9.0/default` | Intel IDB debugger. |
| `idb/9.1/default` | Intel IDB debugger. |
| `ifort/8.0/default` | Intel Fortran Version 8.0 compilers. |
| `ifort/8.1/default` | Intel Fortran Version 8.1 compilers. |
| `ifort/9.0/default` | Intel Fortran Version 9.0 compilers. |
| `ifort/9.1/default` | Intel Fortran Version 9.1 compilers. |
| `imkl/8.0` (default) | Intel Math Kernel Library. |
| `intel/7.1` | Intel Version 7.1 compilers. |
| `intel/8.0` | Intel Version 8.0 compilers. |
| `intel/8.1` | Intel Version 8.1 compilers. |
| `intel/9.0` | Intel Version 9.0 compilers. |
| `intel/9.1` (default) | Intel Version 9.1 compilers. |
| `mlib/intel/7.1` | HP Math Library for Intel 7.1 compilers. |
| `mlib/intel/8.0` | HP Math Library for Intel 8.0 compilers. |
| `mlib/intel/8.1` | HP Math Library for Intel 8.1 compilers. |
| `mlib/intel/9.0` | HP Math Library for Intel 9.0 compilers. |
| `mpi/hp/default` | HP-MPI. |
| `mpi/quadrics/default` | Quadrics MPI |
| `pathscale/2.2/default` | Pathscale 2.2 Debugger |
| `pathscale/2.4/default` | Pathscale 2.4 Debugger |
| `pathscale/2.5/default` | Pathscale 2.5 Debugger |
| `pbspro/default` | PBS Pro batch queuing. |
| `pgi/5.1/default` | PGI Version 5.1 compilers. |
| `pgi/5.2/default` | PGI Version 5.2 compilers. |
| `pgi/6.0/default` | PGI Version 6.0 compilers. |
| `pgi/6.1/default` | PGI Version 6.1 compilers. |
| `pgi/default` (default) | PGI Version 6.0 compilers. |
| `totalview/default` | TotalView debugger. |
| `xtools` | Graphical performance utilities (see Chapter 7 (page 69)) |

See “Viewing Available Modulefiles” for information on how to find which modules are available on your system. See “Viewing Loaded Modulefiles” to determine which of those modules are currently loaded.

Each module supplies its own online help. See "Viewing Modulefile-Specific Help" for information on how to view it.

## 3.3 Modulefiles Automatically Loaded on the System

The HP XC system does not load any modulefiles into your environment by default. However, there may be modulefiles designated by your system administrator that are automatically loaded. "Viewing Loaded Modulefiles" describes how you can determine what modulefiles are currently loaded on your system.

You can also automatically load your own modules by creating a login script and designating the modulefiles to be loaded in the script. You can also add or remove modules from their current environment on a per-module basis, as described in "Loading a Modulefile".

## 3.4 Viewing Available Modulefiles

Available modulefiles are modulefiles that have been provided with the HP XC system software and are available for you to load. A modulefile must be loaded before it provides changes to your environment, as described in the introduction to this section. You can view the modulefiles that are available on the system by issuing the `module avail` command:

```
$ module avail
```

## 3.5 Viewing Loaded Modulefiles

A loaded modulefile is a modulefile that has been explicitly loaded in your environment by the `module load` command. To view the modulefiles that are currently loaded in your environment, issue the `module list` command:

```
$ module list
```

## 3.6 Loading a Modulefile

You can load a modulefile in to your environment to enable easier access to software that you want to use by executing the `module load` command. You can load a modulefile for the current session, or you can set up your environment to load the modulefile whenever you log in to the system.

When loading a modulefile, note that certain modulefiles cannot be loaded while other modulefiles are currently loaded. For example, this can happen with different versions of the same software. If a modulefile you are attempting to load conflicts with a currently loaded modulefile, the modulefile will not be loaded and an error message will be displayed.

If you encounter a modulefile conflict when loading a modulefile, you must unload the conflicting modulefile before you load the new modulefile. See "Modulefile Conflicts" (page 37) for further information about modulefile conflicts.

### 3.6.1 Loading a Modulefile for the Current Session

You can load a modulefile for your current login session as needed. To do this, issue the `module load` command as shown in the following example, which illustrates the `TotalView` modulefile being loaded:

```
$ module load totalview
```

Loading a modulefile in this manner affects your environment for the current session only.

### 3.6.2 Automatically Loading a Modulefile at Login

If you frequently use one or more modulefiles that are not loaded when you log in to the system, you can set up your environment to automatically load those modulefiles for you. A method for doing this is to modify your shell startup script to include instructions to load the modulefile automatically.

For example, if you wanted to automatically load the TotalView modulefile when you log in, edit your shell startup script to include the following instructions. This example uses `bash` as the login shell. Edit the `~/.bashrc` file as follows:

```
# if the 'module' command is defined, $MODULESHOME
# will be set
if [ -n "$MODULESHOME" ]; then
 module load totalview
fi
```

From now on, whenever you log in, the TotalView modulefile is automatically loaded in your environment.

## 3.7 Unloading a Modulefile

In certain cases, you may find it necessary to unload a particular modulefile before you can load another modulefile in to your environment, to avoid modulefile conflicts. See "Modulefile Conflicts" for information about modulefile conflicts.

You can unload a modulefile by using the `module unload` command, as shown in the following example:

```
$ module unload ifort/8.0
```

Unloading a modulefile that is loaded by default makes it inactive for the current session only — it will be reloaded the next time you log in.

## 3.8 Viewing Modulefile-Specific Help

You can view help information for any of the modulefiles on the HP XC system. For example, to access modulefile-specific help information for TotalView, issue the `module help` command as follows:

```
$ module help totalview

----------- Module Specific Help for 'totalview/default' -----------------

This loads the TotalView environment.

Version <default>
Modifies: MANPATH, PATH, TVDSVRLAUNCHCMD
```

**NOTE:** The term `<default>` is displayed whenever the modulefile originator does not use version numbers to distinguish the modulefiles.

## 3.9 Modulefile Conflicts

Some modulefiles should not be loaded while certain other modulefiles are currently loaded. This is especially true of modulefiles for different versions of the same software. For example, the `Intel C/C++ Version 8.0` compiler modulefile should not be loaded while the `Intel C/C++ Version 8.1` compiler modulefile is loaded. A modulefile conflict occurs in this situation.

The system displays an error message when you attempt to load a modulefile that conflicts with one or more currently loaded modulefiles. For example:

```
$ module load ifort/8.0
ifort/8.0(19):ERROR:150: Module 'ifort/8.0' conflicts with the
currently loaded module(s) 'ifort/8.1'

ifort/8.0(19):ERROR:102: Tcl command execution failed:
conflict ifort/8.1
```

In this example, a user attempted to load the `ifort/8.0` modulefile. After the user issued the command to load the modulefile, an error message occurred, indicating a conflict between this modulefile and the `ifort/8.1` modulefile, which is already loaded.

When a modulefile conflict occurs, unload the conflicting modulefile before loading the new modulefile. In the previous example, you should unload the `ifort/8.0` modulefile before loading the `ifort/8.1` modulefile. For information about unloading a modulefile, see "Unloading a Modulefile" (page 37).

**Note:**

To avoid problems, HP recommends that you always unload one version of a modulefile before loading another version.

## 3.10 Creating a Modulefile

If you download or install a software package into a private directory, you can create your own (private) modulefile for products that you install by using the following general steps:

1. Create a private modulefiles directory.
2. Copy an existing modulefile (to use as a template), or copy the software's corresponding default modulefile from under `/opt/modules/modulefiles`, into the private modulefiles directory.
3. Edit and modify the modulefile accordingly.
4. Register the private directory with the `module use` command.

To install a variant of a product or package already on the system, copy the existing modulefile for that product to an appropriate name, and edit it accordingly to accommodate the newly-installed product variant.

To install a random product or package should look at the manpages for modulefiles, examine the existing modulefiles, and create a new modulefile for the product being installed using existing modulefiles as a template. To view modules manpages, load the `modules` modulefile and then display the `modulefile` manpage:

```
$ module load modules
$ man modulefile
```

Read the manpages for modules so that you know how to create a directory for your private modulefiles and how to use the `module use <dirname>` module command to use your private modules.

# 4 Developing Applications

This chapter discusses topics associated with developing applications in the HP XC environment. Before reading this chapter, you should you read and understand Chapter 1 "Overview of the User Environment" and Chapter 2 "Using the System".

This chapter addresses the following topics:

## 4.1 Application Development Environment Overview

The HP XC system provides an application development environment that enables developing, building, and running applications using multiple nodes with multiple cores. These applications can be parallel applications using many cores, or serial applications using a single core.

The HP XC system is made up of nodes that are assigned one or more roles. Nodes with the login role (login nodes) and nodes with the compute role (**compute node**s) are important to the application developer:

- Compute nodes run user applications.
- Login nodes are where you log in and interact with the system to perform such tasks as executing commands, compiling and linking applications, and launching applications. A login node can also execute single-core applications and commands, just as on any other standard Linux system.

Applications are launched from login nodes, and then distributed and run on one or more compute nodes.

The HP XC environment uses the LSF batch job scheduler to launch and manage parallel and serial applications. When a job is submitted, LSF places the job in a queue and allows it to run when the necessary resources become available. When a job is completed, LSF returns job output, job information, and any errors. In addition to batch jobs, LSF can also run interactive batch jobs and interactive jobs. An LSF interactive batch job is a batch job that allows you to interact with the application, yet still take advantage of LSF scheduling policies and features. An LSF interactive job is run without using the batch processing features of LSF, but is dispatched immediately by LSF on the **LSF execution host** node. LSF is described in detail in Chapter 10 "Using LSF".

Regardless of whether an application is parallel or serial, or whether it is run interactively or as a batch job, the general steps to developing an HP XC application are as follows:

1. Build the code by compiling and linking with the correct compiler. Note that compiler selection, and set up of appropriate parameters for specific compilers, is made easier by the use of modules.
2. Launch the application with the `bsub`, `srun`, or `mpirun` command.

The build and launch commands are executed from the node to which you are logged in.

### HP Unified Parallel C Support

HP XC System Software provides support for the HP Unified Parallel C (UPC) application development environment.

HP UPC is a parallel extension of the C programming language, which runs on both common types of multiprocessor systems: those with a common global address space (such as SMP) and those with distributed memory. UPC provides a simple shared memory model for parallel programming, allowing data to be shared or distributed among a number of communicating processors. Constructs are provided in the language to permit simple declaration of shared data, distribute shared data across threads, and synchronize access to shared data across threads. This model promises significantly easier coding of parallel applications and maximum performance across shared memory, distributed memory, and hybrid systems.

See the following Web page for more information about HP UPC:

http://www.hp.com/go/upc

## 4.2 Compilers

You can use compilers acquired from other vendors on an HP XC system. For example, Intel C/C++ and Fortran compilers for the 64-bit architecture and Portland Group C/C++ and Fortran compilers on the CP4000 platform.

Intel, PGI, and Pathscale compilers are not supplied with the HP XC system.

You can use other compilers and libraries on the HP XC system as on any other system, provided they contain single-core routines and have no dependencies on another message-passing system.

Table 4-1 displays the compiler commands for Standard Linux, Intel, and PGI compilers for the C, C++, and Fortran languages.

**Table 4-1 Compiler Commands**

| Type | Compilers | | | Notes |
|---|---|---|---|---|
| | C | C++ | Fortran | |
| Standard Linux | `gcc` | `gcc++` | `g77` | All HP XC platforms.<br>The HP XC System Software supplies these compilers by default. |
| Intel | `icc` | `icc` | `ifort` | Version 9.1compilers<br>For use on the Intel 64–bit platform. |
| Intel | `icc` | `icc` | `ifort` | Version 9.0 compilers<br>For use on the Intel 64–bit platform. |
| Intel | `icc` | `icc` | `ifort` | Version 8.0 compilers<br>For use on the Intel 64–bit platform. |
| Intel | `ecc` | `ecc` | `efc` | Version 7.1 compilers<br>For use on the Intel 64–bit platform.<br>These compilers can be used but Intel may not support them much longer. |
| PGI | `pgcc` | `pgCC` | `pgf95`, `pgf77` | For use on the CP4000 platform |
| Pathscale | `pathcc` | `pathCC` | `pathf95`, `pathf90` | For use on the CP4000 platform<br>`pathf90` provided for backward compatibility. |

### 4.2.1 MPI Compiler

The HP XC System Software includes **MPI**. The MPI library on the HP XC system is HP-MPI Version 2.2.5.1.

## 4.3 Examining Nodes and Partitions Before Running Jobs

Before launching an application, you can determine the availability and status of the system's nodes and partitions. Node and partition information is useful to have before launching a job so that you can launch the job to properly match the resources that are available on the system.

When invoked with no options, the SLURM `sinfo` command returns information about node availability and partitions, along with other information:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE  NODELIST
lsf          up  infinite     4  down*  n[12-15]
slurm*       up   infinite    2  idle   n[10-11]
```

The previous `sinfo` output shows that there are two partitions on the system:

- One for LSF jobs
- One for SLURM jobs

The asterisk in the PARTITION column indicates the default partition. An asterisk in the STATE column indicates nodes that are currently not responding.

See Chapter 9 "Using SLURM" for information about using the `sinfo` command. The SLURM `sinfo` manpage also provides detailed information about the `sinfo` command.

## 4.4 Interrupting a Job

A job launched by the `srun` command can be interrupted by sending a signal to the command by issuing one or more `Ctrl/C` key sequences. Signals sent to the `srun` command are automatically forwarded to the tasks that it is controlling.

The `Ctrl/C` key sequence will report the state of all tasks associated with the `srun` command. If the `Ctrl/C` key sequence is entered twice within one second, the associated `SIGINT` signal will be sent to all tasks. If a third `Ctrl/C` key sequence is entered, the job will be terminated without waiting for remote tasks to exit.

The `Ctrl/Z` key sequence is ignored.

## 4.5 Setting Debugging Options

In general, the debugging information for your application that is needed by most debuggers can be produced by supplying the `-g` switch to the compiler. For more specific information about debugging options, see the documentation and manpages associated with your compiler.

## 4.6 Developing Serial Applications

This section describes how to build and run **serial application**s in the HP XC environment. The following topics are covered:

- "Serial Application Build Environment" (page 42) describes the serial application programming model.
- "Building Serial Applications" (page 42) discusses how to build serial applications.

For further information about developing serial applications, see the following sections:

- "Debugging Serial Applications" (page 63) describes how to debug serial applications.
- "Launching Jobs with the srun Command" (page 81) describes how to launch applications with the `srun` command.
- "Building and Running a Serial Application" (page 115) provides examples of serial applications.

## 4.6.1 Serial Application Build Environment

You can build and run serial applications in the HP XC programming environment. A serial application is a command or application that does not use any form of parallelism.

An example of a serial application is a standard Linux command, such as the `ls` or `hostname` command. A serial application is basically a single-core application that has no communication library calls such as MPI.

## 4.6.2 Building Serial Applications

This section discusses how to build serial applications on an HP XC system. Compiling, linking, and running serial applications are discussed.

To build a serial application, you must be logged in to an HP XC node with the login role. Serial applications are compiled and linked by invoking compilers and linkers.

You launch a serial application either by submitting it to LSF with the `bsub` command, or by invoking the `srun` command to run it. The process is similar to launching a **parallel application**, except that only one compute node core is used. To run on an compute node processor, the serial application and any required dynamic libraries must be accessible from that node. A serial application can also be tested locally by running it on the login node.

### 4.6.2.1 Compiling and Linking Serial Applications

Serial applications are compiled and linked by invoking compile and link drivers.

You can change compilers by using modules. For information about using modules, see "Overview of Modules" (page 33).

As an alternative to using dynamic libraries, serial applications can also be linked to static libraries. Often the `-static` option is used to do this.

For examples of building serial applications with the GNU C, GNU Fortran, and Intel C/C++, and Intel Fortran compilers, see "Building and Running a Serial Application" (page 115).

## 4.7 Developing Parallel Applications

This section describes how to build and run parallel applications. The following topics are discussed:

- "Parallel Application Build Environment" (page 42)
- "Building Parallel Applications" (page 45)

For further information about developing parallel applications in the HP XC environment, see the following:

- "Launching Jobs with the srun Command" (page 81)
- "Debugging Parallel Applications" (page 63)
- Chapter "Advanced Topics" (page 107)

## 4.7.1 Parallel Application Build Environment

This section discusses the **parallel application** build environment on an HP XC system.

The HP XC parallel application environment allows parallel application processes to be started and stopped together on a large number of application cores, along with the I/O and process control structures to manage these kinds of applications.

The HP XC system software uses the HP-MPI distributed memory programming model for building and running parallel applications. In addition to using HP-MPI for parallel application development, OpenMP and Pthreads can be used in conjunction with HP-MPI or separately under the HP XC system software. The section discusses these development tools as they relate to the HP XC system.

#### 4.7.1.1 Modulefiles

The basics of your working environment are set up automatically by your system administrator during the installation of HP XC. However, your application development environment can be modified by means of modulefiles, as described in "Overview of Modules".

There are modulefiles available that you can load yourself to further tailor your environment to your specific application development requirements. For example, the TotalView module is available for debugging applications. "Overview of Modules" provides instructions on how to list what modulefiles are available for you to load, and how load a modulefile.

If you encounter problems accessing tools or commands (and associated manpages), ensure that required modules are loaded on your system. If necessary, load required modules yourself, as described in "Overview of Modules". Otherwise, contact your system administrator.

#### 4.7.1.2 HP-MPI

With the HP XC system software, you can use the HP-**MPI** distributed memory programming model for building and running parallel applications. In this model, all data is private to each process. All intercore communication within a parallel application is performed through calls to the HP-MPI message passing library. Even though support for applications that use a shared-memory programming model is not available at this time, individual cores within an application node can be used in the same application as separate HP-MPI tasks. Applications that are based on MPI, and currently run on Linux (or Linux compatible) systems, can be easily migrated to an HP XC **cluster**.

See the HP-MPI documentation for more information.

#### 4.7.1.3 OpenMP

The OpenMP specification is a set of compiler directives that can be used to specify shared-memory parallelism in Fortran and C/C++ programs. Both Intel and Portland Group Fortran and C/C++ compilers support OpenMP.

Although OpenMP is designed for use on shared-memory architectures, OpenMP can be used on an HP XC system within a node.

OpenMP can be used alone, or in conjunction with HP-MPI. For information about compiling programs using OpenMP, see the OpenMP documentation.

#### 4.7.1.4 Pthreads

POSIX Threads (Pthreads) is a standard library that programmers can use to develop portable threaded applications. Pthreads can be used in conjunction with HP-MPI on the HP XC system. Compilers from GNU, Intel and PGI provide a `-pthread` switch to allow compilation with the Pthread library.

Packages that link against Pthreads, such as MKL, require that the application is linked using the `-pthread` option. The Pthread option is invoked with the following compiler-specific switches:

| | |
|---|---|
| GNU | `-pthread` |
| Intel | `-pthread` |
| PGI | `-lpgthread` |

For example:

```
$ mpicc object1.o ... -pthread -o myapp.exe
```

#### 4.7.1.5 Quadrics SHMEM

The Quadrics implementation of SHMEM runs on HP XC systems with Quadrics switches. SHMEM is a collection of high-performance routines (that support a distributed-memory model) for data passing between parallel executables.

To compile programs that use SHMEM, it is necessary to include the `shmem.h` file and to use the SHMEM and Elan libraries. For example:

```
$ gcc -o shping shping.c -lshmem -lelan
```

#### 4.7.1.6 MPI Library

The **MPI** library supports MPI 1.2 as described in the 1997 release of *MPI: A Message Passing Interface Standard*. Users should note that the MPI specification describes the application programming interface, but does not specify the contents of the MPI header files, `mpi.h` and `mpif.h`, that are included in the source code. Therefore, an MPI application must be recompiled using the proper header files for the MPI library to which it is to be linked.

Parallel applications that use MPI for communication must include the HP XC infrastructure libraries. MPI applications must be built with `mpicc`, `mpic++`, `mpif77`, or `mpif90` utilities.

When an MPI application is launched, the user environment, including any MPI environment variables that have been set, is passed to the application.

MPI profiling support is included in the HP XC MPI library, so you do not need to link with a separate library to access the `PMPI_xxx()` versions of the MPI routines.

The HP XC **cluster** comes with a modulefile for HP-MPI. The `mpi` modulefile is used to set up the necessary environment to use HP-MPI, such as the values of the search paths for header and library files.

#### 4.7.1.7 Intel Fortran and C/C++Compilers

You can use Intel Fortran compilers (Version 7.x and greater) on the HP XC cluster. However, the HP XC cluster does not supply a copy of Intel compilers. Intel compilers must be obtained directly from the vendor. See the Intel documentation for information about using these compilers.

#### 4.7.1.8 PGI Fortran and C/C++ Compilers

You can use PGI Fortran 95, Fortran 77, and C/C++ compilers on the HP XC cluster. However, the HP XC cluster does not supply a copy of PGI compilers. PGI compilers must be obtained directly from the vendor. For information about using the PGI compilers, see the PGI documentation.

#### 4.7.1.9 GNU C and C++ Compilers

You can use the GNU C and C++ compilers on the HP XC cluster. The HP XC cluster supplies copies of the GNU C and C++ compilers.

#### 4.7.1.10 Pathscale Compilers

You can use the Pathscale EKOPath Version 2.1 Compiler Suite on the CP4000 platform only. See the following Web site for more information:

http://www.pathscale.com/ekopath.html.

#### 4.7.1.11 GNU Parallel Make

The GNU parallel Make command is used whenever the `make` command is invoked. GNU parallel Make provides the ability to do a parallel Make; however, all compiling takes place on the login node. Therefore, whether a parallel `make` improves build time depends upon how many cores are on the login node and the load on the login node.

Information about using the GNU parallel Make is provided in "Using the GNU Parallel Make Capability".

For further information about using GNU parallel Make, see the `make` manpage. For additional sources of GNU information, see the references provided in the front of this manual, located in "About This Document".

#### 4.7.1.12 MKL Library

MKL is a math library that references `pthreads`, and in enabled environments, can use multiple threads. MKL can be linked in a single-threaded manner with your application by specifying the following in the link command:

- On the CP3000 and CP4000 platforms (as appropriate):

  ```
  -L/opt/intel/mkl70/lib/32 -lmkl_ia32 -lguide -pthread
  -L/opt/intel/mkl70/lib/em64t -lmkl_em64t -lguide -pthread
  ```

- On the CP6000 platforms:

  ```
  -L/opt/intel/mkl70/lib/64 -lmkl_ipf -lguide -pthread
  ```

#### 4.7.1.13 ACML Library

You can use the AMD Core Math Library (ACML library) on the CP4000 platform.

#### 4.7.1.14 Other Libraries

Other libraries can be used as they would on any other system. However they must contain single core routines and have no dependencies on another message passing system.

### 4.7.2 Building Parallel Applications

This section describes how to build MPI and non-MPI parallel applications on an HP XC system.

#### 4.7.2.1 Compiling and Linking Non-MPI Applications

If you are building non-MPI applications, such as an OpenMP application for example, you can compile and link them on an HP XC as you normally would, with standard header files and switches.

#### 4.7.2.2 Compiling and Linking HP-MPI Applications

This section provides some general information about how to build an HP-MPI application in the HP XC environment.

Compiling and linking an MPI application on an HP XC system is performed by invoking the HP-MPI compiler utilities. HP-MPI compiler utilities are scripts supplied by HP-MPI to make it easier to invoke a compiler with the appropriate libraries and search paths. The HP-MPI compiler utilities add all the necessary path locations and library specifications to the compile and link steps that are required to build a HP XC parallel application. It is highly recommended that you use the HP-MPI compiler utilities to compile and link your MPI application on an HP XC cluster, rather than invoke a compiler directly.

The `mpicc`, `mpic++`, `mpif90`, and `mpif77` MPI compiler commands are used to invoke the HP-MPI compiler utilities that compile and link an MPI application. The `mpicc` and `mpic++` commands invoke the drivers of the C and C++ compilers. The `mpif77` and `mpif90` commands invoke the drivers of the Fortran 77 and Fortran 90 compilers.

Before you can compile and link an MPI program using the MPI compiler commands, the MPI compiler utilities module (`mpi`) must be loaded by using the `module load mpi` command, or you must arrange for them to be in your `$PATH` search list. The use of modules is described in "Overview of Modules".

#### 4.7.2.3 Examples of Compiling and Linking HP-MPI Applications

The following examples show how to compile and link your application code by invoking a compiler utility.

If you have not already loaded the `mpi` compiler utilities module , load it now as follows:

```
$ module load mpi
```

To compile and link a C application using the `mpicc` command:

```
$ mpicc -o mycode hello.c
```

To compile and link a Fortran application using the `mpif90` command:

```
$ mpif90 -o mycode hello.f
```

In the above examples, the HP-MPI commands invoke compiler utilities which call the C and Fortran compilers with appropriate libraries and search paths specified to build the parallel application called `hello`. The `-o` specifies that the resulting program is called `mycode`.

## 4.8 Developing Libraries

This section discusses developing shared and archive libraries for HP XC applications. Building a library generally consists of two phases:

- Compiling sources to objects
- Assembling the objects into a library
  - Using the `ar` archive tool for archive (`.a`) libraries
  - Using the linker (possibly indirectly by means of a compiler) for shared (`.so`) libraries.

For sufficiently small shared objects, it is often possible to combine the two steps.

A common technique is to build the archive library first, and then build the shared library from the archive library (using the linker's `-whole-archive` switch).

For libraries that do not use HP-MPI, it is recommended that the sources be compiled with the standard compilers (such as `gcc`), just as they would be on other UNIX-like platforms.

For libraries that do use HP-MPI, it is possible to use the HP-MPI compiler utilities (such as `mpicc`) to compile the sources to objects. For example:

```
$ mpicc -c -g foo.c
```

To assemble an archive library, use the `ar` archive tool as you would on other UNIX-like platforms. To assemble a shared library, use the linker (possibly indirectly by means of a compiler) as you would on other UNIX-like platforms.

Once the library is built, it can be used to build applications, just as other libraries are used, for both serial applications (with the standard compilers) and parallel applications (with the HP-MPI compiler utilities).

Note that for shared libraries it is necessary to use `LD_LIBRARY_PATH` to include the directory containing the shared library, just as you would on other UNIX-like platforms.

### 4.8.1 Designing Libraries for the CP4000 Platform

This section discusses the issues surrounding the design of libraries for CP4000 platform on the HP XC system.

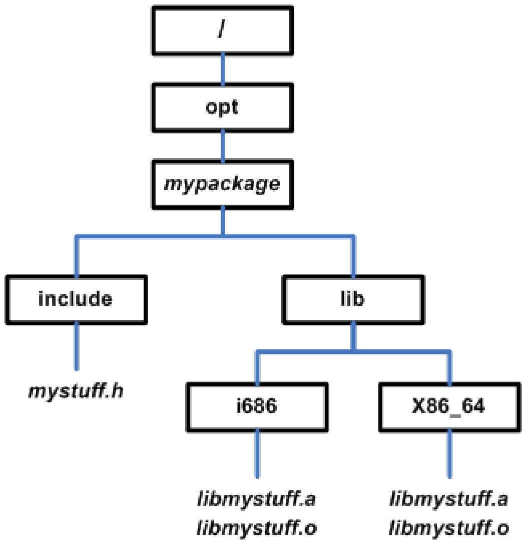
A user designing a library for use on an HP XC CP4000 platform can supply a 32-bit library and/or a 64-bit library. HP recommends both, to provide flexibility and to make it easy to get the 64-bit advantages locally, but be able to take the 32-bit variant to an x86-class machine or run a 32-bit variant imported from an x86-class machine.

It is the library designer's responsibility to make sure 32-bit and 64-bit object files do not collide during the build process. This can be done by "cleaning" object files from the directories between builds, or (as is more common) maintaining separate directories for the different types of objects. Separate directories also makes it easy to maintain production versions distinct from debuggable versions.

Different compilers have different ways to select 32-bit or 64-bit compilations and links. Consult the documentation for the compiler for this information.

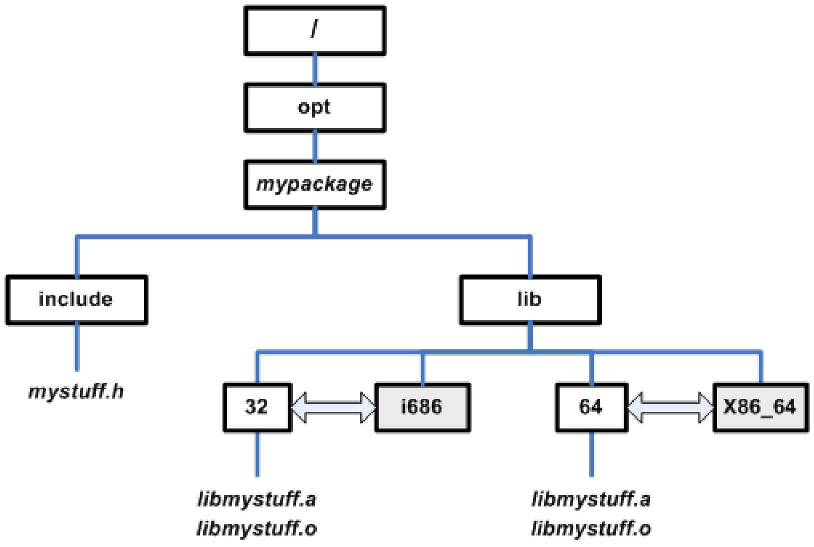
For released libraries, dynamic and archive, the usual custom is to have a `../lib` directory that contains the libraries. This, by itself, will work if the 32-bit and 64-bit libraries have different names. However, HP recommends an alternative method. The dynamic linker, during its attempt to load libraries, will suffix candidate directories with the machine type. The HP XC system on the CP4000 platform uses `i686` for 32-bit binaries and `x86_64` for 64-bit binaries. HP recommends structuring directories to reflect this behavior. Therefore, if your released directory structure has a form similar to Figure 4-1, then ensure that the `LD_LIBRARY_PATH` has `/opt/mypackage/lib` in it, which will then be able to handle both 32-bit and 64-bit binaries that have linked against `libmystuff.so`.

**Figure 4-1 Library Directory Structure**



If you have an existing pattern using different names, HP recommends introducing links with the above names. An example of this is shown in Figure 4-2.

**Figure 4-2 Recommended Library Directory Structure**



Linking an application using the library (dynamic or archive) requires you to specify the appropriate subdirectory, depending on whether the application is 32-bit or 64-bit.

For example, to build a 32-bit application, you might enter:

```
<linkcommand> <32-bit> -L/opt/mypackage/lib/i686 -lmystuff
```

To build a 64-bit application, you might enter:

```
<linkcommand> <64-bit> -L/opt/mypackage/lib/x86_64 -lmystuff
```

**NOTE:** There is no shortcut as there is for the dynamic loader.

# 5 Submitting Jobs

This chapter describes how to submit jobs on the HP XC system; it addresses the following topics:

## 5.1 Overview of Job Submission

On an HP XC system, a job is submitted to LSF, which places the job in a queue and allows it to run when the necessary resources become available. The LSF `bsub` command is the primary method for submitting jobs on the HP XC system.

The format of the `bsub` command depends on the type of the job, as listed here:

- Serial job; that is, a job that runs on a single core
- Non-MPI parallel job
- HP-MPI parallel job
- A batch job script

The remaining sections describe how to submit a job for each of these job types.

The examples in this section submit the `hostname` command or a variation of a `"hello, world"` program. Most examples are run as interactive jobs to display the output.

The examples in this chapter are run on an HP XC system configuration in which `lsfhost.localdomain` is the virtual IP name of the **LSF execution host** and nodes `n[1-16]` are **compute node**s in the `lsf` partition. All nodes contain 2 cores, providing 32 cores for use by LSF jobs.

## 5.2 Submitting a Serial Job Using LSF

There are various methods for submitting a serial job on the HP XC system:

- Using the LSF `bsub` command alone.
- Using the LSF `bsub` command in conjunction with the SLURM `srun` command.
- Using the LSF `bsub` command in conjunction with the SLURM `srun` command and the LSF-SLURM External Scheduler.
- Using the SLURM `srun` command alone.

These methods are explored in this section.

For most instances, the recommended method for submitting a job is to use the LSF `bsub` command in conjunction with the SLURM `srun` command.

### 5.2.1 Submitting a Serial Job with the LSF bsub Command

Use the `bsub` command to submit a serial job on the LSF execution host using the following format:

```
bsub [bsub-options ]  [ srun [srun-options]] jobname [job-options]
```

The `bsub` command launches the job.

Use the `bsub` command's `-I` option to launch an interactive serial job.

The `srun` command is only necessary to launch the job on the allocated node if the HP XC `JOB STARTER` script is not configured to run a job on the compute nodes in the `lsf` partition.

The *`jobname`* parameter can be name of an executable or a batch script. If *`jobname`* is executable, job is launched on LSF execution host node. If *`jobname`* is batch script (containing `srun` commands), job is launched on LSF node allocation (compute nodes). LSF node allocation is created by `-n` *`num-procs`* parameter, which specifies the number of cores the job requests.

The SLURM `srun` job launch command is only needed if the LSF `JOB_STARTER` script is not configured for the intended queue, but it can be used regardless of whether or not the script is configured. You can use the `bqueues` command to confirm whether or not the `JOB_STARTER` script exists; see *bqueues*(1) for information on the `bqueues` command.

Example 5-1 invoking the `bsub` command after which a job can be entered from the standard input.

**Example 5-1 Submitting a Job from the Standard Input**

```
bsub
```

Example 5-2 shows the submission and launch of a serial interactive job and its output.

**Example 5-2 Submitting a Serial Job Using LSF**

```
$ bsub -I srun hostname
Job <20> is submitted to default queue <normal>.
 <<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
```

Example 5-3 shows the submission of an interactive serial job without the SLURM `srun` command and the job output.

**Example 5-3 Submitting an Interactive Serial Job Using LSF only**

```
$ bsub -I hostname
Job <73> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
```

Example 5-4 uses the LSF-SLURM External Scheduler to submit a job to run on four cores on two specific compute nodes.

**Example 5-4 Submitting an Interactive Serial Job Using LSF and the LSF-SLURM External Scheduler**

```
$ bsub -I -n4 -ext "SLURM[nodelist=n[14,16]]" srun hostname
Job <9> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n14
n14
n16
n16
```

### 5.2.2 Submitting a Serial Job Through SLURM Only

This section describes how to build a simple `hello world` application, called `hw_hostname`, execute it on the login node, and launch it with the SLURM `srun` command.

The following is the C source code for this program; the file name is `hw_hostname.c`.

```
#include <unistd.h>
#include <stdio.h>

int main()
{
  char name[100];
  gethostname(name, sizeof(name));
  printf("%s says Hello!\n", name);
  return 0;
}
```

The following is the command line used to compile this program:

```
$ cc hw_hostname.c -o hw_hostname
```

---

**NOTE:** The following invocations of the sample `hw_hostname` program are run on a SLURM non-root default partition, which is not the default SLURM partition for the HP XC system software.

---

When run on the login node, it shows the name of the login node, `n16` in this case:

```
$ ./hw_hostname
n16 says Hello!
```

When you use the `srun` command to submit this program, it runs on one of the compute nodes. In this instance, it runs on node `n13`:

```
$ srun ./hw_hostname
n13 says Hello!
```

Submitting the same program again with the `srun` command may run this program on another node, as shown here:

```
$ srun ./hw_hostname
n12 says Hello!
```

The `srun` can also be used to replicate the program on several cores. Although it is not generally useful, it illustrates the point. Here, the same program is run on 4 cores on 2 nodes.

```
$ srun -n4 ./hw_hostname
n13 says Hello!
n13 says Hello!
n14 says Hello!
n14 says Hello!
```

The output for this command could also have been 1 core on each of 4 compute nodes in the SLURM allocation.

## 5.3 Submitting a Parallel Job

When submitting a parallel job, you can specify the use of HP-MPI. You can also opt to schedule the job by using SLURM. Depending on which submission method you choose, read the appropriate sections, as follows:

- "Submitting a Non-MPI Parallel Job" (page 51)
- "Submitting a Parallel Job That Uses the HP-MPI Message Passing Interface" (page 52)
- "Submitting a Parallel Job Using the SLURM External Scheduler" (page 53)

### 5.3.1 Submitting a Non-MPI Parallel Job

to submit a parallel job

Use the following format of the LSF `bsub` command to submit a parallel job that does not make use of HP-**MPI** to an LSF node allocation (**compute node**s). An LSF node allocation is created by the `-n` *`num-procs`* parameter, which specifies the minimum number of cores the job requests.

```
bsub -n num-procs  [bsub-options] srun [srun-options] jobname [job-options]
```

The `bsub` command submits the job to LSF.

The `-n` *`num-procs`* parameter, which is required for parallel jobs, specifies the number of cores requested for the job. The *`num-procs`* parameter may be expressed as *`minprocs`*`[,`*`maxprocs`*`]` where *`minprocs`* specifies the minimum number of cores and the optional value *`maxprocs`* specifies the maximum number of cores.

The SLURM `srun` command is required to run jobs on an LSF node allocation. The `srun` command is the user job launched by the LSF `bsub` command. SLURM launches the *`jobname`* in parallel on the reserved cores in the `lsf` partition.

The *`jobname`* parameter is the name of an executable file or command to be run in parallel.

Example 5-5 illustrates a non-MPI parallel job submission. The job output shows that the job "`srun hostname`" was launched from the LSF execution host `lsfhost.localdomain`, and that it ran on 4 cores from the compute nodes `n1` and `n2`.

**Example 5-5 Submitting a Non-MPI Parallel Job**

```
$ bsub -n4 -I srun hostname
Job <21> is submitted to default queue <normal>
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n1
n2
n2
```

You can use the LSF-SLURM external scheduler to specify additional SLURM options on the command line. As shown in Example 5-6, it can be used to submit a job to run one task per compute node (on **SMP** nodes):

**Example 5-6 Submitting a Non-MPI Parallel Job to Run One Task per Node**

```
$ bsub -n4 -ext "SLURM[nodes=4]" -I srun hostname
Job <22> is submitted to default queue <normal>
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n2
n3
n4
```

## 5.3.2 Submitting a Parallel Job That Uses the HP-MPI Message Passing Interface

Use the following format of the LSF `bsub` command to submit a parallel job that makes use of HP-MPI:

`bsub -n` *`num-procs`* `[`*`bsub-options`*`]` *`mpijob`*

The `bsub` command submits the job to LSF.

The `-n` *`num-procs`* parameter, which is required for parallel jobs, specifies the number of cores requested for the job.

The *`mpijob`* argument has the following format:

`mpirun` [*`mpirun-options`*] [`-srun`] [*`srun-options`*] [*`mpi-jobname`*] [*`job-options`*]

See the *mpirun*(1) manpage for more information on this command.

The `mpirun` command's `-srun` option is required if the `MPI_USESRUN` environment variable is not set or if you want to use additional `srun` options to execute your job.

The `srun` command, used by the `mpirun` command to launch the **MPI** tasks in parallel in the `lsf` partition, determines the number of tasks to launch from the `SLURM_NPROCS` environment variable that was set by LSF; this environment variable is equivalent to the number provided by the `-n` option of the `bsub` command.

Any additional SLURM `srun` options are job specific, not allocation-specific.

The *`mpi-jobname`* is the executable file to be run. The *`mpi-jobname`* must be compiled with the appropriate HP-MPI compilation utility. For more information, see the section titled *Compiling applications* in the *HP-MPI User's Guide*.

Example 5-7 shows an MPI job that runs a `hello world` program on 4 cores on 2 compute nodes.

**Example 5-7 Submitting an MPI Job**

```
$ bsub -n4 -I mpirun -srun ./hello_world
Job <24> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
Hello world!
Hello world!  I'm 1 of 4 on host1
Hello world!  I'm 3 of 4 on host2
Hello world!  I'm 0 of 4 on host1
Hello world!  I'm 2 of 4 on host2
```

You can use the LSF-SLURM External Scheduler option to add capabilities at the job level and queue level by including several SLURM options in the command line. For example, you can use it to submit a job to run one task per node, or to submit a job to run on specific nodes. "LSF-SLURM External Scheduler" discusses this option.

Example 5-8 shows an MPI job that uses the LSF-SLURM External Scheduler option to run the same `hello world` program on each of 4 compute nodes.

**Example 5-8 Submitting an MPI Job with the LSF-SLURM External Scheduler Option**

```
$ bsub -n4 -ext "SLURM[nodes=4]" -I mpirun -srun ./hello_world
Job <27> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
Hello world!
Hello world!  I'm 1 of 4 on host1
Hello world!  I'm 2 of 4 on host2
Hello world!  I'm 3 of 4 on host3
Hello world!  I'm 4 of 4 on host4
```

Some preprocessing may need to be done:

- If the **MPI** job requires the use of an *`appfile`*
- If there is another reason that prohibits the use of the `srun` command as the task launcher

This preprocessing should determine the node host names to which `mpirun`'s standard task launcher should launch the tasks. In such scenarios, you need to write a batch script; there are several methods available for determining the nodes in an allocation. One method is to use the `SLURM_JOBID` environment variable with the `squeue` command to query the nodes. Another method is to use the LSF environment variables such as `LSB_HOSTS` and `LSB_MCPU_HOSTS`, which are prepared by the HP XC job starter script.

### 5.3.3 Submitting a Parallel Job Using the SLURM External Scheduler

The LSF-SLURM external scheduler option provides additional capabilities at the job level and queue level by allowing the inclusion of several SLURM options in the LSF command line. With LSF integrated with SLURM, you can use the LSF-SLURM External Scheduler to specify SLURM options that specify the minimum number of nodes required for the job, specific nodes for the job, and so on.

**Note:**

The SLURM external scheduler is a plug-in developed by Platform Computing for LSF; it is not actually part of SLURM. This plug-in communicates with SLURM to gather resource information and request allocations of nodes, but it is integrated with the LSF scheduler.

The format of this option is shown here:

```
-ext "SLURM[slurm-arguments]"
```

The `bsub` command format to submit a parallel job to an LSF allocation of compute nodes using the external scheduler option is as follows:

```
bsub -n num-procs -ext "SLURM[slurm-arguments]" [bsub-options]
[ -srun [srun-options]
] [jobname] [job-options]
```

The *`slurm-arguments`* parameter can be one or more of the following `srun` options, separated by semicolons, as described in Table 5-1.

**Table 5-1 Arguments for the SLURM External Scheduler**

| SLURM Arguments | Function |
|---|---|
| `nodes=`*`min`*`[-`*`max`*`]` | Specifies the minimum and maximum number of nodes allocated to job. The job allocation will contain at least the minimum number of nodes. |
| `mincpus=`*`ncpus`* | Specifies minimum number of cores per node. Default value is 1. |
| `mem=`*`value`* | Specifies a minimum amount of real memory in megabytes of each node. |
| `tmp=`*`value`* | Specifies a minimum amount of temporary disk space in megabytes of each node. |
| `constraint=`*`feature`* | Specifies a list of constraints. The list may include multiple features separated by "&" or "\|". "&" represents ANDed, "\|" represents ORed. |
| `nodelist=`*`list of nodes`* | Requests a specific list of nodes. The job will at least contain these nodes. The list may be specified as a comma-separated list of nodes or a range of nodes |
| `exclude=`*`list of nodes`* | Requests that a specific list of hosts not be included in resource allocated to this job. The list may be specified as a comma-separated list of nodes or a range of nodes. |
| `contiguous=yes` | Requests a mandatory contiguous range of nodes. |

The Platform LSF documentation provides more information on general external scheduler support.

Consider an HP XC system configuration in which `lsfhost.localdomain` is the LSF execution host and nodes `n[1-10]` are compute nodes in the `lsf` partition. All nodes contain two cores, providing 20 cores for use by LSF jobs.

Example 5-9 shows one way to submit a parallel job to run on a specific node or nodes.

## Example 5-9 Using the External Scheduler to Submit a Job to Run on Specific Nodes

```
$ bsub -n4 -ext "SLURM[nodelist=n6,n8]" -I srun hostname
Job <70> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n6
n6
n8
n8
```

In the previous example, the job output shows that the job was launched from the LSF execution host `lsfhost.localdomain`, and it ran on four cores on the specified nodes, `n6` and `n8`.

Example 5-10 shows one way to submit a parallel job to run one task per node.

## Example 5-10 Using the External Scheduler to Submit a Job to Run One Task per Node

```
$ bsub -n4 -ext "SLURM[nodes=4]" -I srun hostname
Job <71> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n2
n3
n4
```

In the previous example, the job output shows that the job was launched from the LSF execution host `lsfhost.localdomain`, and it ran on four cores on four different nodes (one task per node).

Example 5-11 shows one way to submit a parallel job to avoid running on a particular node.

## Example 5-11 Using the External Scheduler to Submit a Job That Excludes One or More Nodes

```
$ bsub -n4 -ext "SLURM[nodes=4; exclude=n3]" -I srun hostname
Job <72> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n2
n4
n5
```

This example runs the job exactly the same as in Example 5-10 "Using the External Scheduler to Submit a Job to Run One Task per Node", but additionally requests that node `n3` is not to be used to run the job. Note that this command could have been written to exclude additional nodes.

Example 5-12 launches the `hostname` command once on nodes `n1` through `n10` (`n[1-10]`):

## Example 5-12 Using the External Scheduler to Launch a Command in Parallel on Ten Nodes

```
$ bsub -n 10 -ext "SLURM[nodelist=n[1-10]]" srun hostname
```

Example 5-13 launches the `hostname` command on 10 cores on nodes with a `dualcore` SLURM feature assigned to them:

**Example 5-13 Using the External Scheduler to Constrain Launching to Nodes with a Given Feature**

```
$ bsub -n 10 -ext "SLURM[constraint=dualcore]" -I srun hostname
```

You can use the bqueues command to determine the SLURM scheduler options that apply to jobs submitted to a specific LSF queue, for example:

```
$ bqueues -l dualcore | grep SLURM
MANDATORY_EXTSCHED:  SLURM[constraint=dualcore]
```

## 5.4 Submitting a Batch Job or Job Script

Use the following bsub command format to submit a batch job or job script:

`bsub -n` *`num-procs`* `[`*`bsub-options`*`]` *`script-name`*

The `-n` *`num-procs`* parameter, which is required for parallel jobs, specifies the number of cores the job requests.

The *`script-name`* argument is the name of the batch job or script. The script can contain one or more `srun` or `mpirun` commands.

The script will execute once on the first allocated node, and the `srun` or `mpirun` commands within the script will be run on the allocated compute nodes.

In Example 5-14, a simple script named `myscript.sh`, which contains two `srun` commands, is displayed then submitted.

**Example 5-14 Submitting a Job Script**

```
$ cat myscript.sh
#!/bin/sh
srun hostname mpirun -srun hellompi
$ bsub -I -n4 myscript.sh
Job <29> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n2
n2
n4
n4
Hello world!  I'm 0 of 4 on  n2
Hello world!  I'm 1 of 4 on  n2
Hello world!  I'm 2 of 4 on  n4
Hello world!  I'm 3 of 4 on  n4
```

Example 5-15 runs the same script but uses the LSF-SLURM External Scheduler option to specify different resources (here, 4 compute nodes).

**Example 5-15 Submitting a Batch Script with the LSF-SLURM External Scheduler Option**

```
$ bsub -n4 -ext "SLURM[nodes=4]" -I ./myscript.sh
Job <79> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n2
n3
n4
Hello world!  I'm 0 of 4 on  n1
Hello world!  I'm 1 of 4 on  n2
Hello world!  I'm 2 of 4 on  n3
Hello world!  I'm 3 of 4 on  n4
```

Example 5-16 and Example 5-17 show how the jobs inside the script can be manipulated within the allocation.

**Example 5-16 Submitting a Batch Job Script That Uses a Subset of the Allocation**

```
$ bsub -n4 -ext "SLURM[nodes=4]" -I ./myscript.sh
Job <80> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n2
Hello world! I'm 0 of 2 on n1
Hello world! I'm 1 of 2 on n2
```

**Example 5-17 Submitting a Batch job Script That Uses the srun --overcommit Option**

```
$ bsub -n4 -I ./myscript.sh
Job <81> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n1
n1
n1
n1
n2
n2
n2
n2
Hello world! I'm 0 of 8 on n1
Hello world! I'm 1 of 8 on n1
Hello world! I'm 2 of 8 on n1
Hello world! I'm 3 of 8 on n1
Hello world! I'm 4 of 8 on n2
Hello world! I'm 5 of 8 on n2
Hello world! I'm 6 of 8 on n2
Hello world! I'm 7 of 8 on n2
```

Example 5-18 shows some of the environment variables that are available in a batch script. The `LSB_HOSTS` and `LSB_MCPU_HOSTS` environment variables are defined in *Platform LSF Reference*. The `SLURM_JOBID` and `SLURM_NPROCS` environment variables are defined in the *SLURM Reference Manual*.

**Example 5-18 Environment Variables Available in a Batch Job Script**

```
$ cat ./envscript.sh
#!/bin/sh
name=`hostname`
echo "hostname = $name"
echo "LSB_HOSTS = '$LSB_HOSTS'"
echo "LSB_MCPU_HOSTS = '$LSB_MCPU_HOSTS'"
echo "SLURM_JOBID = $SLURM_JOBID"
echo "SLURM_NPROCS = $SLURM_NPROCS"
$ bsub -n4 -I ./envscript.sh
Job <82> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
hostname = n1
LSB_HOSTS = 'n1 n1 n2 n2'
LSB_MCPU_HOSTS = 'n1 2 n2 2'
SLURM_JOBID = 176
SLURM_NPROCS = 4
```

## 5.5 Submitting Multiple MPI Jobs Across the Same Set of Nodes

There are two ways to run multiple MPI jobs across the same set of nodes at the same time; they are:

- Using a script
- Using a Makefile

The following sections show these methods. In both methods, the jobs submitted are parallel jobs using the HP-MPI message passing interface and use the `ping_pong_ring` program, which is delivered with the HP-MPI software.

### 5.5.1 Using a Script to Submit Multiple Jobs

You can write a script that consists of multiple commands that launch jobs. In this example, the `ping_pong_ring` command is run first in the background then again in the foreground:

```
$ cat script
#!/bin/sh
mpirun -srun -N2 -n4 ./ping_pong_ring 100000 &
mpirun -srun -N2 -n4 ./ping_pong_ring 100000
```

The following command line executes the script, which submits the jobs:

```
$ bsub -o %J.out -n2 -ext "SLURM[nodes=2]" ./script
Job <111> is submitted to default queue <normal>.
```

The `bjobs` command provides information on the execution of the `script`:

```
$ bjobs
JOBID   USER    STAT  QUEUE    FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
111     lsfadmi PEND  normal   lsfhost.loc             ./script   date and time
```

Use the `squeue` command to acquire information on the jobs:

```
$ squeue -s
STEPID   NAME           PARTITION USER       TIME NODELIST
13.0     hptclsf@111          lsf lsfadmin   0:07 n14
13.1     ping_pong_ring       lsf lsfadmin   0:07 n[14-15]
13.2     ping_pong_ring       lsf lsfadmin   0:07 n[14-15]
$ bjobs
No unfinished job found
```

### 5.5.2 Using a Makefile to Submit Multiple Jobs

You can submit multiple jobs across the same nodes by using a Makefile.

For information on `Makefiles` and the `make` utility, see *The GNU Make Manual* and *make*(1L).

The ping_pong_ring application is submitted twice in a Makefile named mymake; the first time as run1 and the second as run2:

```
$ cat mymake

  PPR_ARGS=10000
  NODES=2
  TASKS=4

  all: run1 run2

  run1:
         mpirun -srun -N ${NODES} -n ${TASKS} ./ping_pong_ring ${PPR_ARGS}

  run2:
         mpirun -srun -N ${NODES} -n ${TASKS} ./ping_pong_ring ${PPR_ARGS}
```

The following command line makes the program and executes it:

```
$ bsub -o %J.out -n2 -ext "SLURM[nodes=2]" make -j2 -f ./mymake
PPR_ARGS=1000000
Job <113> is submitted to default queue <normal>.
```

Use the squeue command to acquire information on the jobs:

```
$ squeue -s
STEPID   NAME           PARTITION USER      TIME NODELIST
15.0     hptclsf@113          lsf lsfadmin  0:04 n14
15.1     ping_pong_ring       lsf lsfadmin  0:04 n[14-15]
15.2     ping_pong_ring       lsf lsfadmin  0:04 n[14-15]
```

The following command displays the final ten lines of the output file generated by the execution of the application made from mymake:

```
$ tail 113.out
1000000 bytes: 937.33 MB/sec
[2:n15] ping-pong 1000000 bytes ...
1000000 bytes: 1048.41 usec/msg
1000000 bytes: 953.82 MB/sec
[3:n15] ping-pong 1000000 bytes ...
1000000 bytes: 15308.02 usec/msg
1000000 bytes: 65.33 MB/sec
[3:n15] ping-pong 1000000 bytes ...
1000000 bytes: 15343.11 usec/msg
1000000 bytes: 65.18 MB/sec
```

The following illustrates how an error in the Makefile is reported. This Makefile specifies a nonexistent program:

```
$ cat mymake

PPR_ARGS=10000
NODES=2
TASKS=4

all: run1 run2 run3

run1:
     mpirun -srun -N ${NODES} -n ${TASKS} ./ping_pong_ring ${PPR_ARGS}

run2:
     mpirun -srun -N ${NODES} -n ${TASKS} ./ping_pong_ring ${PPR_ARGS}

run3:
     mpirun -srun -N ${NODES} -n ${TASKS} ./ping_bogus ${PPR_ARGS} 1
```

1 This line attempts to submit a program that does not exist.

The following command line makes the program and executes it:

```
$ bsub -o %J.out -n2 -ext "SLURM[nodes=2]" make -j3 \
-f ./mymake PPR_ARGS=100000
Job <117> is submitted to default queue <normal>.
```

The output file contains error messages related to the attempt to launch the nonexistent program.

```
$ cat 117.out

   .
   .
   .

mpirun -srun -N 2 -n 4 ./ping_pong_ring 100000
mpirun -srun -N 2 -n 4 ./ping_pong_ring 100000
mpirun -srun -N 2 -n 4 ./ping_bogus 100000
slurmstepd: [19.3]: error: execve(): ./ping_bogus: No such file or directory
slurmstepd: [19.3]: error: execve(): ./ping_bogus: No such file or directory
srun: error: n14: task0: Exited with exit code 2
srun: Terminating job
slurmstepd: [19.3]: error: execve(): ./ping_bogus: No such file or directory
slurmstepd: [19.3]: error: execve(): ./ping_bogus: No such file or directory
make: *** [run3] Error 2
make: *** Waiting for unfinished jobs....
[0:n14] ping-pong 100000 bytes ...
100000 bytes: 99.06 usec/msg
100000 bytes: 1009.51 MB/sec
[0:n14] ping-pong 100000 bytes ...
100000 bytes: 99.76 usec/msg
100000 bytes: 1002.43 MB/sec
[1:n14] ping-pong 100000 bytes ...
100000 bytes: 1516.83 usec/msg
100000 bytes: 65.93 MB/sec
[1:n14] ping-pong 100000 bytes ...
100000 bytes: 1519.73 usec/msg
100000 bytes: 65.80 MB/sec
[2:n15] ping-pong 100000 bytes ...
100000 bytes: 108.65 usec/msg
100000 bytes: 920.38 MB/sec
[2:n15] ping-pong 100000 bytes ...
100000 bytes: 99.44 usec/msg
100000 bytes: 1005.65 MB/sec
[3:n15] ping-pong 100000 bytes ...
100000 bytes: 1877.35 usec/msg
100000 bytes: 53.27 MB/sec
[3:n15] ping-pong 100000 bytes ...
100000 bytes: 1888.22 usec/msg
100000 bytes: 52.96 MB/sec
```

The `sacct` command, which displays SLURM accounting information, reflects the error:

```
  [lsfadmin@n16 ~]$ sacct -j 19
  Jobstep    Jobname            Partition    Ncpus Status     Error
  ---------- ------------------ ---------- ------- ---------- -----
  19         hptclsf@117        lsf              8 CANCELLED      2
  19.0       hptclsf@117        lsf              0 FAILED         2
  19.1       hptclsf@117        lsf              8 COMPLETED      0
  19.2       hptclsf@117        lsf              8 COMPLETED      0
  19.3       hptclsf@117        lsf              8 FAILED         2
```

## 5.6 Submitting a Job from a Host Other Than an HP XC Host

To submit a job from a host other than an HP XC host to the HP XC system, use the LSF `-R` option, and the HP XC host type `SLINUX64` (defined in `lsf.shared`) in the job submission resource requirement string. The necessary resource requirement string to submit a job from a host other than an HP XC host is specified as follows:

```
-R "type=SLINUX64"
```

The following example shows this resource requirement string in an LSF command:

```
$ bsub -R "type=SLINUX64"  -n4 -I srun hostname
```

## 5.7 Running Preexecution Programs

A *preexecution program* is a program that performs needed setup tasks that an application needs. It may create directories, input files, and so on.

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

Where preexecution commands run depends on the type of job:

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

Before starting a preexecution program, LSF sets the `SLURM_JOBID` environment variable. To enable `srun` to launch preexecution on the first allocated node and other allocated nodes, your preexecution program should pick up the `SLURM_JOBID` environment variable. The `SLURM_JOBID` has the information LSF needs to run the job on the nodes required by your preexecution program.

The following items provide the information you need to run the preexecution program on the resource manager node, on the first allocated node, or on all the allocated nodes:

| | |
|---|---|
| To run a preexecution program on the resource manager node: | This is the default behavior. Run the preexecution program normally. Your preexecution does not need to make use of the `SLURM_JOBID` environment variable. |
| To run a preexecution program on the first allocated node: | Use the SLURM `srun -N 1` command. For example: `$ /opt/hptc/bin/srun -N 1 my_pre_exec` |
| To run a preexecution program on all allocated nodes: | Use the SLURM `srun` directly without node options. For example: `$ /opt/hptc/bin/srun  my_pre_exec` |

**NOTE:** Do not use the `srun -b` command (for SLURM batch mode) inside preexecution programs. The `srun -b` command returns immediately after a SLURM batch job is submitted. This can cause the preexecution program to exit with success while the real task is still running in batch mode.

See the SLURM *srun*(1) manpage for more information about the `srun` command, its options, and the `SLURM_JOBID` environment variable.

# 6 Debugging Applications

This chapter describes how to debug serial and parallel applications in the HP XC development environment. In general, effective debugging of applications requires the applications to be compiled with debug symbols, typically the `-g` switch. Some compilers allow `-g` with optimization.

This chapter addresses the following topics:

## 6.1 Debugging Serial Applications

You can debug a **serial application** on an HP XC system with the same method you would use to debug a serial application on a conventional Linux operating system. For information about debugging serial programs, see the standard Linux documentation.

The following serial debuggers are available for use in the HP XC environment for local debugging:

- The `gdb` utility is provided with the standard Linux distribution; it performs line-mode debugging of a single process.
- The `idb` utility is generally available with the Intel compiler suite.
- The `pgdbg` utility is generally available with the PGI compilers.

For information about using these debuggers, see the standard Linux documentation and the documentation that is available with the specific debugger that you are using.

## 6.2 Debugging Parallel Applications

The following parallel debuggers are recommended for use in the HP XC environment are TotalView and DDT.

### TotalView

TotalView is a full-featured GUI debugger for debugging parallel applications from Etnus, Inc. It is specifically designed to meet the requirements of parallel applications running on many cores. The use of TotalView in the HP XC environment is described in "Debugging with TotalView". You can obtain additional information about TotalView from the TotalView documentation and the TotalView Web site at:

http://www.etnus.com

**Note:**

TotalView is not included with the HP XC software and is not supported. If you have any problems installing or using TotalView, contact Etnus, Inc.

### DDT

DDT (Distributed Debugging Tool) is a parallel debugger from Streamline Computing. DDT is a comprehensive graphical debugger designed for debugging parallel code. It gives users a common interface for most compilers, languages and **MPI** distributions. For information about using DDT, see the Streamline Computing documentation and the Streamline Computing Web site:

http://www.streamline-computing.com/softwaredivision_1.shtml

## 6.2.1 Debugging with TotalView

TotalView is a full-featured, debugger based on GUI and specifically designed to fill the requirements of parallel applications running on many cores.

You can purchase the TotalView debugger, from Etnus, Inc., for use on the HP XC **cluster**.

TotalView is not included with the HP XC software and technical support is not provided by HP. Contact Etnus, Inc. for any issues with TotalView.

This section provides only minimum instructions to get you started using TotalView. Instructions for installing TotalView are included in the *HP XC System Software Installation Guide*. Read the TotalView documentation for full information about using TotalView; the TotalView documentation set is available directly from Etnus, Inc. at the following URL:

http://www.etnus.com

### 6.2.1.1 SSH and TotalView

As discussed in "Using the Secure Shell to Log In" and "Enabling Remote Execution with OpenSSH", HP XC systems use the OpenSSH package in place of traditional commands like `rsh` to provide more secure communication between nodes in the cluster. When run in a parallel environment, TotalView expects to be able to use the `rsh` command to communicate with other nodes, but the default HP XC configuration disallows this.

Set the `TVDSVRLAUNCHCMD` environment variable to specify an alternate command for TotalView to use in place of `rsh`. When using the TotalView Modulefile, as described in "Setting Up TotalView", this variable is automatically set to `/usr/bin/ssh -o BatchMode=yes`. If you manage your environment independently of the provided modulefiles, set this variable manually.

### 6.2.1.2 Setting Up TotalView

TotalView must be set up as described here:

1. Determine if TotalView is installed, and whether environment variables have been defined for TotalView. Use the `which` or `whereis` to do so.
2. Determine if environment variables have been defined for TotalView. You can use the `echo` for this.
3. Set the `DISPLAY` environment variable of the system that hosts TotalView to display on your local system.

   Also, run the `xhosts` command locally to accept data from the system that hosts TotalView; see the *X*(7X) manpage for more information.

4. Edit the `PATH` environment variable to include the location of the TotalView executable.

   Also add the location of the TotalView manpages to the `MANPATH` environment variable.

   The following list summarizes some suggestions:

   - Edit your login file or profile file to include the following commands:

     ```
     module load mpi
     module load totalview
     ```

   - Set the `PATH` and `MANPATH` environment variables in your shell initialization file or `login` file.
   - Have your system administrator set up your environment so that the TotalView modulefile loads automatically when you log in to the system.
   - Adjust your environment manually before invoking TotalView.

   See "Overview of Modules" for information on modulefiles.

   Your administrator may have already installed TotalView and set up the environment for you. In this case, skip the steps in this section and proceed to "Setting TotalView Preferences", which describes using TotalView for the first time.

### 6.2.1.3 Using TotalView with SLURM

Use the following commands to allocate the nodes you need before you debug an application with SLURM, as shown here:

```
$ srun -Nx -A
$ mpirun -tv -srun application
```

These commands allocate *x* nodes and run TotalView to debug the program named `application`.

Be sure to exit from the SLURM allocation created with the `srun` command when you are done.

### 6.2.1.4 Using TotalView with LSF

HP recommends the use of `xterm` when debugging an application with LSF. You also need to allocate the nodes you will need.

You may need to verify the full path name of the `xterm` and `mpirun` commands:

First run a `bsub` command to allocate the nodes you will need and to launch an `xterm` window:

```
$ bsub -nx -ext "SLURM[nodes=x]" \
-Is /usr/bin/xterm
```

Enter an `mpirun -tv` command in the `xterm` window to start TotalView on the application you want to debug:

```
$ mpirun -tv -srun application
```

### 6.2.1.5 Setting TotalView Preferences

You should set TotalView preferences the first time you invoke it. For example, you need to tell TotalView how to launch TotalView processes on all the cores.

The TotalView preferences are maintained in a preferences file named `.totalview/preferences` in your home directory, so you only need to set these preferences once.

Follow these steps to set the TotalView preferences:

1. Invoke the TotalView debugger:

   ```
   $ totalview
   ```

   TotalView's main control window (called the *TotalView Root Window*) appears.

2. Select `Preferences` from the `File` pull-down menu.

   A `Preferences` window opens.

3. Select (that is, click on) the `Launch Strings` tab in the `Preferences` window.
4. Ensure that the `Enable single debug server launch` button is selected in the `Launch Strings` tab.
5. In the `Launch Strings` table, in the area immediately to the right of `Command:`, verify that the default command launch string shown is as follows:

   ```
   %C %R -n "%B/tvdsvr -working_directory %D -callback %L -set_pw %P
   -verbosity %V %F"
   ```

   You may be able to obtain this setting by selecting the `Defaults` button; otherwise, you need to enter this command launch string.

6. Select the `Bulk Launch` tab in the `Preferences` window.

   Make sure that `Enable debug server bulk launch` is *not* selected.

7. Select the `OK` button at the bottom-left of the `Preferences` window to save these changes.
8. Exit TotalView by selecting `Exit` from the `File` pull-down menu.

The TotalView launch preferences are configured and saved. You can change this configuration at any time.

### 6.2.1.6 Debugging an Application

This section describes how to use TotalView to debug an application.

1. Compile the application to be debugged. For example:

   ```
   $ mpicc -g -o Psimple simple.c -lm
   ```

   Use the `-g` option to enable debugging information.

2. Run the application in TotalView:

   ```
   $ mpirun -tv -srun -n2 ./Psimple
   ```

3. The TotalView main control window, called the *TotalView root window*, opens. It displays the following message in the window header:

   ```
   Etnus TotalView Version#
   ```

4. The *TotalView process window* opens.

   This window contains multiple panes that provide various debugging functions and debugging information. The name of the application launcher that is being used (either `srun` or `mpirun`) is displayed in the title bar.

5. Set the search path if you are invoking TotalView from a directory that does not contain the executable file and the source code. If TotalView is invoked from the same directory, you can skip to step Step 6.

   Set the search path as follows:

   a. Select the `File` pull-down menu of the TotalView process window.
   b. Select `Search Path` from the list that appears.

   TotalView, by default, will now search for source and binaries (including symbol files) in the following places and in the following order:

   - Current working directory
   - Directories in `File` ⇒ `Search Path`
   - Directories specified in your PATH environment variable

6. Select the `Go` button in the TotalView process window. A pop-up window appears, asking if you want to stop the job:

   ```
   Process srun is a parallel job.
   Do you want to stop the job now?
   ```

7. Select `Yes` in this pop-up window. The TotalView root window appears and displays a line for each process being debugged.

   If you are running Fortran code, another pop-up window may appear with the following warning:

   ```
   Sourcefile initfdte.f was not found, using assembler mode.
   ```

   Select `OK` to close this pop-up window. You can safely ignore this warning.

8. You can now set a breakpoint somewhere in your code. The method to do this may vary slightly between versions of TotalView. For TotalView Version 6.0, the basic process is as follows:

   a. Select `At Location` in the `Action Point` pull-down menu of the TotalView process window.
   b. Enter the name of the location where you want to set a breakpoint.
   c. Select `OK`.

9. Select the `Go` button to run the application and go to the breakpoint.

Continue debugging as you would on any system. If you are not familiar with TotalView, you can select on `Help` in the right-hand corner of the process window for additional information.

### 6.2.1.7 Debugging Running Applications

As an alternative to the method described in “Debugging an Application”, it is also possible to "attach" an instance of TotalView to an application which is already running.

1. Compile a long-running application as in “Debugging an Application”:

   ```
   $ mpicc -g -o Psimple simple.c -lm
   ```

2. Run the application:

   ```
   $ mpirun -srun -n2 Psimple
   ```

3. Start TotalView:

   ```
   $ totalview
   ```

4. Select `Unattached` in the TotalView Root Window to display a list of running processes.

   Double-click on the `srun` process to attach to it.

5. The TotalView Process Window appears, displaying information on the `srun` process.

   Select `Attached` in the TotalView Root Window.

6. Double-click one of the remote `srun` processes to display it in the TotalView Process Window.
7. Now you should be able set breakpoints to debug the application.

### 6.2.1.8 Exiting TotalView

Make sure your job has completed before exiting TotalView. This may require that you wait a few seconds from the time your job has completed until `srun` has completely exited.

If you exit TotalView before your job is completed, use the `squeue` command to ensure that your job is not still on the system.

```
$ squeue
```

If it is still there, use the following command to remove all of your jobs:

```
$ scancel --user username
```

To cancel individual jobs, see the `scancel` manpage for information about selective job cancellation.

# 7 Monitoring Node Activity

This chapter describes the optional utilities that provide performance information about the set of nodes associated with your jobs. It addresses the following topics:

- "The Xtools Utilities" (page 69)
- "Running Performance Health Tests" (page 70)

## 7.1 The Xtools Utilities

Two Xtools utilities help you monitor your HP XC system:

| | |
|---|---|
| `xcxclus` | Enables you to monitor a number of nodes simultaneously. |
| `xcxperf` | Enables you to monitor the performance a single node for a variety of metrics. |

You can invoke these utilities with an option to store their data so that you can play back the display. You also can generate a plot file with which you can plot the data.

The `xcxclus` and `xcxperf` Xtools utilities are not installed by default. Have your system administrator install the following `xtools` RPMs using the instructions in the *HP XC System Software Administration Guide*:

- `xtools-common-`*`version.platform`*`.rpm`
- `xtools-xc_clients-`*`version.platform`*`.rpm`

where:

*`version`* Identifies the RPM version.

*`platform`* Refers to the platform:

- `ia64` denotes the CP6000 platform
- `x86_64` denotes the CP3000 and CP4000 platforms

The *HPCPI and Xtools Version 0.6.6 User's Guide* located at the following web address contains complete instructions on using the `xcxclus` and `xcxperf` utilities:

http://docs.hp.com/en/5992-4009/HPCPI_and_Xtools_User_Gd.pdf

**NOTE:** The *HPCPI and Xtools Version 0.6.6 User's Guide* describes other utilities that are not provided in the HP XC system distribution. Only the information on the `xcxclus` and `xcxperf` utilities applies.

The Xtools utilities described in the *HPCPI and Xtools Version 0.6.6 User's Guide* have generic and enhanced displays. By default, the Xtools RPM`xtools-common` and `xtools-xc_clients` RPMs enable only the generic features (that is, options and displays) for use with the HP XC System Software. The enhanced features require a separate software package containing a third Xtools RPM and an RPM for HPCPI. If you want the versatility and capability that the enhanced options and displays offer, HP requires that you complete a simple and separate license agreement for the components that enable them. To access that license agreement, or for more information, contact:

Mr. Jim Bovay **`jim.bovay@hp.com`**

Hewlett-Packard Co
Attn: Jim Bovay III
Mailstop MRO1-1/M8
200 Forest Street
Marlborough, MA 01752-3085
United States

## 7.2 Running Performance Health Tests

You can run the `ovp` command to generate reports on the performance health of the nodes. Use the following format to run a specific performance health test:

`ovp [`*`options`*`] [-verify=perf_health/`*`test`*`]`

Where:

*`options`* Specify additional command line options for the test. The `ovp --help perf_health` command lists the command line options for each test.

The following options apply to all the tests:

**NOTE:** Use the `--opts=` option to pass this option.

- The `--nnodes=`*`n`* option runs the test on *`n`* compute nodes.
- The `--nodelist=`*`nodelist`* option specifies the compute nodes to use.

  **NOTE:** The `--nodelist=`*`nodelist`* option is particularly useful for determining problematic nodes.

  If you use this option and the `--nnodes=`*`n`* option, the `--nnodes=`*`n`* option is ignored.
- The `--queue` *LSF_queue* option specifies the LSF queue for the performance health tests.

*`test`* Indicates the test to perform. The following tests are available:

| | |
|---|---|
| `cpu` | Tests CPU core performance using the Linpack benchmark. |
| `cpu_usage` | Tests CPU core usage. All CPU cores should be idle during the test. This test reports a node if it is using more than 10% (by default) of its CPU cores.<br>The head node is excluded from this test. |
| `memory` | Uses the streams benchmark to test memory performance. |
| `memory_usage` | Tests memory usage. This test reports a node that uses more than 25 percent (by default) of its memory. |
| `network_stress` | Tests network performance. Check network performance under stress using the Pallas benchmark's Alltoall, Allgather, and Allreduce tests. These tests should be performed on a large number of nodes for the most accurate results.<br>The default value for the number of nodes is 4, which is the minimum value that should be used.<br>The `--all_group` option allows you to select the node grouping size. |
| `network_bidirectional` | Tests network performance between pairs of nodes using the Pallas benchmark's Exchange test. |
| `network_unidirectional` | Tests network performance between pairs of nodes using the HP MPI `ping_pong_ring` test. |

**NOTE:** Except for the `network_stress` and `network_bidirectional` tests, these tests only apply to systems that install LSF incorporated with SLURM. The `network_stress` and `network_bidirectional` tests also function under Standard LSF.

You can list the available tests with the `ovp -l` command:

```
$ ovp -l

Test list for perf_health:

    cpu_usage

    memory_usage

    cpu

    memory

    network_stress

    network_bidirectional

    network_unidirectional
```

By default, the `ovp` command reports if the nodes passed or failed the given test. Use the `ovp --verbose` option to display additional information.

The results of the test are written to a file in your home directory. The file name has the form `ovp_`*`node`*`_`*`date`*`[rx].log` where *`node`* is the node from which the command was launched and *`date`* is a date stamp in the form *mmddyy*. Subsequent test runs of the test are identified with an `r` (for run) and a number; this prevents a log file from overwriting a previous one. The `--chdir=` option enables you to designate a directory different from your home directory as the execution directory.

The following is an example of the CPU usage test:

```
$ ovp --verify=perf_health/cpu_usage

XC CLUSTER VERIFICATION PROCEDURE
date time

Verify perf_health:

    Testing cpu_usage ...

        +++ PASSED +++


This verification has completed successfully.

A total of 1 test was run.


Details of this verification have been recorded in:

        HOME_DIRECTORY/ovp_n16_mmddyy.log
```

The following example runs the same test but with the `--verbose` option to show additional output.

```
$ ovp --verbose --verify=perf_health/cpu_usage
XC CLUSTER VERIFICATION PROCEDURE
date time
```

```
Verify perf_health:

    Testing cpu_usage ...

        The headnode is excluded from the cpu usage test.
        Number of nodes allocated for this test is 14
        Job <102> is submitted to default queue <interactive>.
        <<Waiting  for dispatch ...>>
        <<Starting on lsfhost.localdomain>>>

        All nodes have cpu usage less than 10%.

        +++ PASSED +++


This verification has completed successfully.

A total of 1 test was run.


Details of this verification have been recorded in:

        HOME_DIRECTORY/ovp_n16_mmddyy.log
```

The following example tests the memory of nodes `n11`, `n12`, `n13`, `n14`, and `n15`. The `--keep` option preserves the test data in a temporary directory.

```
$ ovp --verbose --opts=--nodelist=n[11-15] --keep \
-verify=perf_health/memory

XC CLUSTER VERIFICATION PROCEDURE
date time

Verify perf_health:

    Testing memory ...

        Specified nodelist is n[11-15]
        Number of nodes allocated for this test is 5
        Job <103> is submitted to default queue <interactive>.
        <<Waiting  for dispatch ...>>
        <<Starting on lsfhost.localdomain>>>

        Detailed streams results for each node can be found in
        path_name
        if the --keep flag was specified.

        Streams memory results summary (all values in mBytes/sec):
          min:    1272.062500
          max:    2090.885900
          median: 2059.492300
          mean:   1865.301018
          range:       818.823400
          variance:    128687.956603
          std_dev:     358.731037

        All nodes were in range for this test.

        +++ PASSED +++


This verification has completed successfully.

A total of 1 test was run.
```

```
The tests execution directory has been saved in:

        HOME_DIRECTORY/ovp_n16_mmddyy.tests

Details of this verification have been recorded in:

        HOME_DIRECTORY/ovp_n16_mmddyyr1.log
```

# 8 Tuning Applications

This chapter discusses how to tune applications in the HP XC environment.

## 8.1 Using the Intel Trace Collector and Intel Trace Analyzer

This section describes how to use the Intel Trace Collector (ITC) and Intel Trace Analyzer (ITA) with HP-MPI on an HP XC system. The Intel Trace Collector/Analyzer were formerly known as VampirTrace and Vampir, respectively. This section addresses the following topics:

- "Building a Program — Intel Trace Collector and HP-MPI" (page 75)
- "Running a Program – Intel Trace Collector and HP-MPI" (page 76)
- "Using the Intel Trace Collector and Intel Trace Analyzer" (page 75)
- "Visualizing Data – Intel Trace Analyzer and HP-MPI" (page 79)

The following Intel Trace Collector/Analyzer documentation is available on the HP XC system after Intel Trace Collector/Analyzer is installed:

- *Intel Trace Collector Users Guide* — located on the HP XC system at:

  `<install-path-name>/ITC/doc/Intel_Trace_Collector_Users_Guide.pdf`.
- *Intel Trace Analyzer Users Guide* — located on the HP XC system at:

  `<install-path-name>/ITA/doc/Intel_Trace_Analyzer_Users_Guide.pdf`

### 8.1.1 Building a Program — Intel Trace Collector and HP-MPI

HP-MPI is MPICH compatible if you use the HP-MPI MPICH scripts which are located at: `/opt/hpmpi/bin`. The HP-MPI MPICH scripts are:

- `mpicc` is replaced by `mpicc.mpich`
- `mpif77` is replaced by `mpif77.mpich`
- `mpirun` is replaced by `mpirun.mpich`

In summary, `mpi`*`xx`* becomes `mpi`*`xx`*`.mpich`.

For further information, see the *Intel Trace Collector Users Guide*. This document is located on the HP XC system at the following location:
`<install-path-name>/ITC/doc/Intel_Trace_Collector_Users_Guide.pdf`.

**Example 8-1 The vtjacobic Example Program**

For the purposes of this example, the examples directory under `/opt/IntelTrace/ITC` is copied to the user's home directory and renamed to `examples_directory`.

The GNU Makefile looks as follows:

```
CC       = mpicc.mpich
F77      = mpif77.mpich
CLINKER  = mpicc.mpich
FLINKER  = mpif77.mpich
IFLAGS   = -I$(VT_ROOT)/include
CFLAGS   = -g
FFLAGS   = -g
LIBS     = -lvtunwind -ldwarf -lnsl -lm -lelf -lpthread
CLDFLAGS = -static-libcxa -L$(VT_ROOT)/lib $(TLIB) -lvtunwind \
           -ldwarf -lnsl -lm -lelf -lpthread
FLDFLAGS = -static-libcxa -L$(VT_ROOT)/lib $(TLIB) -lvtunwind \
           -ldwarf -lnsl -lm -lelf -lpthread
```

In the cases where Intel compilers are used, add the `-static-libcxa` option to the link line. Otherwise the following type of error will occur at run-time:

```
$ mpirun.mpich -np 2 ~/examples_directory/vtjacobic
~/examples_directory/vtjacobic:
    error while loading shared libraries:
libcprts.so.6: cannot open shared object file:
    No such file or directory

MPI Application rank 0 exited before MPI_Init() with status 127
mpirun exits with status: 127
```

In “The vtjacobic Example Program”, `~/examples_directory/vtjacobic` represents the file specification for the `vtjacobic` example program.

For more information, see the following Web site:

http://support.intel.com/support/performancetools/c/linux/sb/CS-010097.htm

## 8.1.2 Running a Program – Intel Trace Collector and HP-MPI

Assuming that you have built your program using the `-static-libcxa` option, as discussed in “Building a Program — Intel Trace Collector and HP-MPI”, you can launch the program with the `mpirun.mpich` command, as shown in the following example.

Example 8-2 shows how to run the `vtjacobic` example program discussed in “Building a Program — Intel Trace Collector and HP-MPI”.

**Example 8-2 C Example – Running the vtjacobic Example Program**

```
$ mpirun.mpich -np 2 ~/examples_directory/vtjacobic
~/examples_directory/vtjacobic:100 iterations
in 0.228252 secs (28.712103 MFlops), m=130 n=130 np=2 [0]
Intel Trace Collector INFO: Writing tracefile vtjacobic.stf
in ~/examples_directory/vtjacobic

mpirun exits with status: 0
```

In Example 8-2, `~/examples_directory/vtjacobic` represents the file specification for the `vtjacobic` example program.

## 8.2 The Intel Trace Collector and Analyzer with HP-MPI on HP XC

**NOTE:** The Intel Trace Collector (OTA) was formerly known as VampirTrace. The Intel Trace Analyzer was formerly known as Vampir.

### 8.2.1 Installation Kit

The following are installation-related notes:

There are two installation kits for the Intel Trace Collector:

- `ITC-IA64-LIN-MPICH-PRODUCT.4.0.2.1.tar.gz`
- `ITA-IA64-LIN-AS21-PRODUCT.4.0.2.1.tar.gz`

The Intel Trace Collector is installed in the `/opt/IntelTrace/ITC` directory.

The Intel Trace Analyzer is installed in the `/opt/IntelTrace/ITA` directory.

The license file is located in the `/opt/IntelTrace/` directory so both tools can find it.

### 8.2.2 HP-MPI and the Intel Trace Collector

This section contains information about building a program with OTA.

HP-MPI is MPICH-compatible if you use the following HP-MPI MPICH scripts, which are located in the `/opt/mpi/bin` directory:

- `mpicc` is replaced by `mpicc.mpich`
- `mpif77` is replaced by `mpif77.mpich`
- `mpirun` is replaced by `mpirun.mpich`

In summary: `mpi`*`XX`* becomes `mpi`*`XX`*`.mpich`

As an example, the `examples` directory under `/opt/IntelTrace/ITC` was copied to a home directory and renamed to `ITC_examples_xc6000`. The GNU makefile `makefile` now resembles the following:

```
CC         = mpicc.mpich
F77        = mpif77.mpich
CLINKER    = mpicc.mpich
FLINKER    = mpif77.mpich
IFLAGS     = -I$(VT_ROOT)/include
CFLAGS     = -g
FFLAGS     = -g
LIBS       =  -lvtunwind -ldwarf -lnsl -lm -lelf -lpthread
CLDFLAGS   = -static-libcxa  -L$(VT_ROOT)/lib  $(TLIB) -lvtunwind -ldwarf
-lnsl -lm -lelf -lpthread
FLDFLAGS   = -static-libcxa -L$(VT_ROOT)/lib  $(TLIB) -lvtunwind -ldwarf
-lnsl -lm -lelf -lpthread
```

When you use the Intel compilers, add `-static-libcxa` to the link line; otherwise, the following errors are generated at runtime:

```
[n1]/nis.home/sballe/xc_PDE_work/ITC_examples_xc6000
>mpirun.mpich -np 2 ~/xc_PDE_work/ITC_examples_xc6000/vtjacobic
warning: this is a development version of HP-MPI for internal R&D use only
/nis.home/sballe/xc_PDE_work/ITC_examples_xc6000/vtjacobic:
error while loading shared libraries: libcprts.so.6:
cannot open shared object file: No such file or directory MPI Application
rank 0 exited before MPI_Init() with status 127 mpirun exits with
status: 127 [n1]/nis.home/sballe/xc_PDE_work/ITC_examples_xc6000 >
```

For specific information on the error `Error While Loading Shared Libraries: libcprts.so.6: cannot open shared object`, see the following URL:

http://support.intel.com/support/performancetools/c/linux/sb/CS-010097.htm

## Running a Program

Ensure that the `-static-libcxa` flag is used when you use `mpirun.mpich` to launch a C or Fortran program.

The following is a C example called `vtjacobic`:

```
# mpirun.mpich -np 2 ~/xc_PDE_work/ITC_examples_xc6000/vtjacobic
warning: this is a development version of HP-MPI for internal R&D use only
/nis.home/user_name/xc_PDE_work/ITC_examples_xc6000/vtjacobic: 100 iterations in
0.228252 secs (28.712103 MFlops), m=130 n=130 np=2
[0] Intel Trace Collector INFO: Writing tracefile vtjacobic.stf in
/nis.home/user_name/xc_PDE_work/ITC_examples_xc6000
mpirun exits with status: 0
```

The following is a Fortran example called `vtjacobif`:

```
# mpirun.mpich -np 2 ~/xc_PDE_work/ITC_examples_xc6000/vtjacobif
warning: this is a development version of HP-MPI for internal R&D use only
           2  Difference is   0.390625000000000
           4  Difference is   0.123413085937500
           6  Difference is   6.341743469238281E-002
           8  Difference is   3.945139702409506E-002
          10  Difference is   2.718273504797253E-002
          12  Difference is   1.992697400677912E-002
          14  Difference is   1.520584760276139E-002
          16  Difference is   1.192225932086809E-002
          18  Difference is   9.527200632430437E-003
          20  Difference is   7.718816241067778E-003
          22  Difference is   6.318016878920021E-003
          24  Difference is   5.211741863535576E-003
          26  Difference is   4.324933536667125E-003
          28  Difference is   3.605700997191797E-003
          30  Difference is   3.016967266492488E-003
          32  Difference is   2.531507385360385E-003
          34  Difference is   2.128864474525351E-003
          36  Difference is   1.793360334698915E-003
          38  Difference is   1.512772311960036E-003
          40  Difference is   1.277430422527046E-003
          42  Difference is   1.079587281504879E-003
          44  Difference is   9.129693228356968E-004
          46  Difference is   7.724509294615510E-004
          48  Difference is   6.538134083283944E-004
          50  Difference is   5.535635456297787E-004
          52  Difference is   4.687947140887371E-004
          54  Difference is   3.970788908277634E-004
          56  Difference is   3.363815146174385E-004
          58  Difference is   2.849935053418584E-004
          60  Difference is   2.414763917353628E-004
          62  Difference is   2.046176064805586E-004
          64  Difference is   1.733937801556939E-004
          66  Difference is   1.469404085386082E-004
          68  Difference is   1.245266549586746E-004
          70  Difference is   1.055343296682637E-004
          72  Difference is   8.944029434752290E-005
          74  Difference is   7.580169395426893E-005
          76  Difference is   6.424353519703476E-005
          78  Difference is   5.444822123484475E-005
          80  Difference is   4.614672291984789E-005
          82  Difference is   3.911112299221254E-005
          84  Difference is   3.314831465581266E-005
          86  Difference is   2.809467246160129E-005
          88  Difference is   2.381154327036583E-005
          90  Difference is   2.018142964565221E-005
          92  Difference is   1.710475838933507E-005
          94  Difference is   1.449714388058985E-005
          96  Difference is   1.228707004052045E-005
          98  Difference is   1.041392661369357E-005
```

```
[0] Intel Trace Collector INFO: Writing tracefile vtjacobif.stf in
/nis.home/user_name/xc_PDE_work/ITC_examples_xc6000
mpirun exits with status: 0
```

### Running a Program Across Nodes (Using LSF)

The following is an example that uses the LSF `bsub` command to run the program named `vtjacobic` across four nodes:

```
# bsub -n4 -I mpirun.mpich -np 2 ./vtjacobic
```

The license file and the OTC directory need to be distributed across the nodes.

## 8.3 Visualizing Data – Intel Trace Analyzer and HP-MPI

The Intel Trace Analyzer is used in a straightforward manner on an HP XC system, as described in its standard documentation. For more information, see the *Intel Trace Analyzer Users Guide*. This document is located on the HP XC system at the following location:

```
<install-path-name>/ITA/doc/Intel_Trace_Analyzer_Users_Guide.pdf
```

# 9 Using SLURM

HP XC uses the Simple Linux Utility for Resource Management (SLURM) for system resource management and job scheduling.

This chapter addresses the following topics:

## 9.1 Introduction to SLURM

SLURM is a reliable, efficient, open source, fault-tolerant, job and compute resource manager with features that make it suitable for large-scale, high performance computing environments. SLURM can report on machine status, perform partition management, job management, and job scheduling.

The *SLURM Reference Manual* is available on the HP XC Documentation CD-ROM and from the following web address:

https://computing.llnl.gov/linux/slurm/documentation.html.

SLURM manpages are also available online on the HP XC system.

As a system resource manager, SLURM has the following key functions:

- Allocate exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work
- Provide a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes
- Arbitrate conflicting requests for resources by managing a queue of pending work

"How LSF and SLURM Interact" describes the interaction between SLURM and LSF.

## 9.2 SLURM Utilities

You interact with SLURM through its command line utilities. The basic utilities are listed here:

- `srun`
- `squeue`
- `scancel`
- `sinfo`
- `scontrol`

For more information on any of these utilities, see the *SLURM Reference Manual* or the corresponding manpage.

## 9.3 Launching Jobs with the srun Command

The `srun` command submits and controls jobs that run under SLURM management. The `srun` command is used to submit interactive and batch jobs for execution, allocate resources, and initiate job steps.

The `srun` command handles both serial and parallel jobs.

The `srun` command has a significant number of options to control the execution of your application closely. However, you can use it for a simple launch of a serial program, as Example 9-1 shows.

**Example 9-1 Simple Launch of a Serial Program**

```
$ srun hostname n1
```

## 9.3.1 The srun Roles and Modes

The `srun` command submits jobs to run under SLURM management. The `srun` command can perform many roles in launching and managing your job. The `srun` command operates in several distinct usage modes to accommodate the roles it performs.

### 9.3.1.1 The srun Roles

The options of the `srun` command allow you control a SLURM job by:

- Specifying the parallel environment for your job when you submit it, such as the number of nodes to use, partition, distribution of processes among nodes, and maximum time.
- Controlling the behavior of your parallel job as it runs, such as by redirecting or labeling its output, sending it signals, or specifying its reporting verbosity.

### 9.3.1.2 The srun Modes

The `srun` command has five distinct **modes** in which it can be used:

- Simple mode
- Batch mode
- Allocate mode
- Attach mode
- Batch (with LSF) mode

The *SLURM Reference Manual* describes the Simple, Batch, Allocate, and Attach modes.

You can submit a script to LSF that contains (simple) `srun` commands to execute parallel jobs later. In this case, LSF takes the place of the `srun -b` option for indirect, across-machine job-queue management.

## 9.3.2 Using the srun Command with HP-MPI

The `srun` command can be used as an option in an HP-MPI launch command. See Chapter Chapter 5: Submitting Jobs for information about using `srun` with HP-MPI.

## 9.3.3 Using the srun Command with LSF

The `srun` command can be used in an LSF launch command. See Chapter Chapter 10: Using LSF for information about using `srun` with LSF.

## 9.4 Monitoring Jobs with the squeue Command

The `squeue` command displays the queue of running and waiting jobs (or "job steps"), including the JobID used for `scancel`), and the nodes assigned to each running job. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

Example 9-2 reports on job 12345 and job 12346:

**Example 9-2 Displaying Queued Jobs by Their JobIDs**

```
$ squeue --jobs 12345,12346
        JOBID PARTITION NAME USER ST TIME_USED NODES NODELIST(REASON)
        12345     debug job1 jody  R      0:21     4 n[9-12]
        12346     debug job2 jody PD      0:00     8
```

The `squeue` command can report on jobs in the job queue according to their state; possible states are: `pending`, `running`, `completing`, `completed`, `failed`, `timeout`, and `node_fail`. Example 9-3 uses the `squeue` command to report on failed jobs.

**Example 9-3 Reporting on Failed Jobs in the Queue**

```
$ squeue --state=FAILED
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
     59      amt1 hostname     root   F       0:00      0
```

## 9.5 Terminating Jobs with the scancel Command

The `scancel` command cancels a pending or running job or job step. It can also be used to send a specified signal to all processes on all nodes associated with a job. Only job owners or administrators can cancel jobs.

Example 9-4 terminates job #415 and all its jobsteps.

**Example 9-4 Terminating a Job by Its JobID**

```
$ scancel 415
```

Example 9-5 cancels all pending jobs.

**Example 9-5 Cancelling All Pending Jobs**

```
$ scancel --state=PENDING
```

Example 9-6 sends the `TERM` signal to terminate jobsteps 421.2 and 421.3.

**Example 9-6 Sending a Signal to a Job**

```
$ scancel --signal=TERM 421.2 421.3
```

## 9.6 Getting System Information with the sinfo Command

The `sinfo` command reports the state of partitions and nodes managed by SLURM. It has a wide variety of filtering, sorting, and formatting options. The `sinfo` command displays a summary of available partition and node (not job) information, such as partition names, nodes/partition, and cores/node.

**Example 9-7 Using the sinfo Command (No Options)**

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     3  down* n[0,5,8]
lsf          up  infinite    14  idle  n[1-4,6-7,9-16]
```

The node STATE codes in these examples may be appended by an asterisk character (`*`) ; this indicates that the reported node is not responding. See the *sinfo*(1) manpage for a complete listing and description of STATE codes.

**Example 9-8 Reporting Reasons for Downed, Drained, and Draining Nodes**

```
$ sinfo -R
      REASON                                NODELIST
      Memory errors                         n[0,5]
      Not Responding                        n8
```

## 9.7 Job Accounting

HP XC System Software provides an extension to SLURM for job accounting. The `sacct` command displays job accounting data in a variety of forms for your analysis. Job accounting data is stored in a log file; the `sacct` command filters that log file to report on your jobs, jobsteps, status, and errors. See your system administrator if job accounting is not configured on your system.

By default, only the superuser is allowed to access the job accounting data. To grant all system users read access to this data, the superuser must change the permission of the `jobacct.log` file, as follows:

```
# chmod a+r /hptc_cluster/slurm/job/jobacct.log
```

You can find detailed information on the `sacct` command and job accounting data in the *sacct*(1) manpage.

## 9.8 Fault Tolerance

SLURM can handle a variety of failure modes without terminating workloads, including crashes of the node running the SLURM controller. User jobs may be configured to continue execution despite the failure of one or more nodes on which they are executing. The command controlling a job may detach and reattach from the parallel tasks at any time. Nodes allocated to a job are available for reuse as soon as the job(s) allocated to that node terminate. If some nodes fail to complete job termination in a timely fashion because of hardware or software problems, only the scheduling of those tardy nodes will be affected.

## 9.9 Security

SLURM has a simple security model:

- Any user of the system can submit jobs to execute. Any user can cancel his or her own jobs. Any user can view SLURM configuration and state information.
- Only privileged users can modify the SLURM configuration, cancel other users' jobs, or perform other restricted activities. Privileged users in SLURM include `root` users and `SlurmUser` (as defined in the SLURM configuration file).

If permission to modify SLURM configuration is required by others, `set-uid` programs may be used to grant specific permissions to specific users.

SLURM accomplishes security by means of communication authentication, job authentication, and user authorization.

For further information about SLURM security features, see the SLURM documentation.

# 10 Using LSF

The Load Sharing Facility (LSF) from Platform Computing is a batch system resource manager used on the HP XC system.

On an HP XC system, a job is submitted to LSF, which places the job in a queue and allows it to run when the necessary resources become available. In addition to launching jobs, LSF provides extensive job management and information capabilities. LSF schedules, launches, controls, and tracks jobs that are submitted to it according to the policies established by the HP XC site administrator.

Two types of LSF are available for installation on the HP XC:

- LSF

  This product is the popular batch system produced by Platform Computing that has become an industry standard. For full information about LSF, see the standard LSF documentation set, which is described in the "Related Software Products and Additional Publications" section of this manual. LSF manpages are also available online on the HP XC system.

- LSF integrated with SLURM (LSF)

  This product is the LSF product from Platform Computing that has been integrated with SLURM to take advantage of SLURM's scalable, efficient resource management and parallel job support.

The intended primary use of the HP XC system determined which of these LSF products was installed.

This chapter introduces you to LSF integrated with SLURM in the HP XC environment. It provides an overview of how LSF integrated with SLURM works, and discusses some of the features and differences of LSF compared to LSF integrated with SLURM on an HP XC system.

This chapter also contains an important discussion of how LSF and SLURM work together to provide the HP XC job management environment. A description of SLURM is provided in Chapter 9 (page 81).

This chapter addresses the following topics:

## 10.1 Information for LSF

The information for LSF is provided in the LSF documentation. This documentation is on the HP XC installation disk and manpages are online.

LSF is installed and configured on all nodes of the HP XC system by default. Nodes without the `compute` **role** are closed with `'0'` job slots available for use.

The LSF environment is set up automatically for the user on login; LSF commands and their manpages are readily accessible:

- The `bhosts` command is useful for viewing LSF batch host information.
- The `lshosts` command provides static resource information.
- The `lsload` command provides dynamic resource information.
- The `bsub` command is used to submit jobs to LSF.
- The `bjobs` command provides information on batch jobs.

## 10.2 Overview of LSF Integrated with SLURM

LSF was integrated with SLURM for the HP XC system to merge the scalable and efficient resource management of SLURM with the extensive scheduling capabilities of LSF. In this integration:

- SLURM manages the compute resources.
- LSF performs the job management.

SLURM extends the parallel capabilities of LSF with its own fast parallel launcher (which is integrated with HP-MPI), full parallel I/O and signal support, and parallel job accounting capabilities.

Managing the compute resources of the HP XC system with SLURM means that the LSF daemons run only on one HP XC node and can present the HP XC system as a single LSF host. As a result:

- All the nodes are configured as LSF Client Hosts; every node is able to access LSF. You can submit jobs from any node in the HP XC system.
- The `lshosts` and `bhosts` commands only list one host that represents all the resources of the HP XC system.

LSF integrated with SLURM obtains resource information about the HP XC system. This information is consolidated and key information such as the total number of cores and the maximum memory available on all nodes becomes the characteristics of the single HP XC “LSF Execution Host”. Additional resource information from SLURM, such as pre-configured node “features”, are noted and processed during scheduling through the external SLURM scheduler for LSF.

Integrating LSF with SLURM on HP XC systems provides you with a parallel launch command to distribute and manage parallel tasks efficiently. The SLURM `srun` command offers much flexibility for requesting requirements across an HP XC system; for example, you can request

- Request contiguous nodes
- Execute only one task per node
- Request nodes with specific features

This flexibility is preserved in LSF through the external SLURM scheduler; this is discussed in more detail in the section titled “LSF-SLURM External Scheduler” (page 92)

A SLURM partition named `lsf` is used to manage LSF jobs. Thus:

- You can view information about this partition with the `sinfo` command.
- The total number of cores listed by the `lshosts` and `bhosts` commands for that host should be equal to the total number of cores assigned to the SLURM `lsf` partition.

When a job is submitted and the resources are available, LSF creates a properly-sized SLURM allocation and adds several standard LSF environment variables to the environment in which the job is to be run. The following two environment variables are also added:

`SLURM_JOBID` This environment variable is created so that subsequent `srun` commands make use of the SLURM allocation created by LSF for the job. This variable can be used by a job script to query information about the SLURM allocation, as shown here:

```
$ squeue --jobs $SLURM_JOBID
```

"Translating SLURM and LSF JOBIDs" describes the relationship between the SLURM_JOBID and the LSF JOBID.

| | |
|---|---|
| SLURM_NPROCS | This environment variable passes along the total number of tasks requested with the bsub -n command to all subsequent srun commands. User scripts can override this value with the srun -n command, but the new value must be less than or equal to the original number of requested tasks. |

LSF regards the entire HP XC system as a "SLURM machine." LSF gathers resource information from SLURM and creates SLURM allocations for each job. As a consequence, every LSF job has a corresponding SLURM JOBID.

For a parallel job, LSF allocates multiple nodes for the job, but LSF always runs the batch script (or user command) on the first node. The batch script or the user command must start its tasks in parallel. The srun command is the SLURM "parallel launcher" command. HP-MPI uses the srun command through the mpirun -srun option.

**Example 10-1 Examples of LSF Job Launch**

The following individual examples are run on a 4-node cluster with 2 cores per nodes:

```
[lsfadmin@n16 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
lsf          up   infinite     4   idle n[13-16]
```

This command line requests 4 cores, but runs the hostname command on the first node:

```
[lsfadmin@xc19n16 ~]$ bsub -n4 -I hostname
Job <110> is submitted to default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
n13
```

The following command line requests 4 cores and uses the srun to run the hostname command on all four:

```
[lsfadmin@n16 ~]$ bsub -n4 -I srun hostname
Job <111> is submitted to default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
n13
n13
n14
n14
```

The following command line requests 4 cores across all 4 nodes and runs the hostname command on each node:

```
[lsfadmin@n16 ~]$ bsub -n4 -I -ext "SLURM[nodes=4]" srun hostname
Job <112> is submitted to default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
n13
n14
n15
n16
```

It is possible to set your SSH keys to avoid password prompting so that you can use SSH-based parallel launchers like the pdsh and mpirun commands. Use the LSB_HOSTS environment variable to pass the list of allocated nodes to the launcher.

**Example 10-2 Examples of Launching LSF Jobs Without the srun Command**

The following `bsub` command line invokes the `bash` shell to run the `hostname` command with the `pdsh` command:

```
[lsfadmin@n16 ~]$ bsub -n4 -I -ext "SLURM[nodes=4]" /bin/bash -c 'pdsh -w "$LSB_HOSTS" hostname'
Job <118> is submitted to default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
n15: n15
n14: n14
n16: n16
n13: n13
```

The following command line uses the `mpirun` command to launch the `hello_world` example program on one core on each of four nodes:

```
[lsfadmin@n16 ~]$ bsub -n4 -I -ext "SLURM[nodes=4]" mpirun -lsb_hosts ./hello_world
Job <119> is submitted to default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
Hello world! I'm 0 of 4 on n13
Hello world! I'm 1 of 4 on n14
Hello world! I'm 2 of 4 on n15
Hello world! I'm 3 of 4 on n16
[lsfadmin@n16 ~]$"
```

The differences described in HP XC System Software documentation take precedence over descriptions in the LSF documentation from Platform Computing. See “Differences Between LSF and LSF Integrated with SLURM” and the *lsf_diff*(1) manpage for more information on the subtle differences between LSF and LSF integrated with SLURM.

## 10.3 Differences Between LSF and LSF Integrated with SLURM

LSF integrated with SLURM for the HP XC environment supports all the standard features and functions that LSF supports, except for those items described in this section, in “Using LSF Integrated with SLURM in the HP XC Environment”, and in the HP XC release notes for LSF.

- By LSF standards, the HP XC system is a single host. Therefore, all LSF “per-host” configuration and “per-host” options apply to the entire HP XC system.

  LSF integrated with SLURM knows only about the HP XC compute nodes through SLURM, so any preference or resource request that is intended for the HP XC compute nodes must go through LSF's external SLURM scheduler. See the “LSF-SLURM External Scheduler” (page 92) for more details.

- LSF requires LSF daemons on every node. These daemons allow LSF to extract detailed information from each node, which you display or use for scheduling. This information includes CPU load, number of users, free memory, and so on.

  When LSF is integrated with SLURM, it runs daemons only on one node in the HP XC system. Therefore, it relies on SLURM for static resource information (that is, number of CPUs, total physical memory, and any assigned SLURM “features”), and bases its scheduling on that static information.

- LSF integrated with SLURM does not collect the following information from each node in the HP XC system:

  - `maxswap`
  - `ndisks`
  - `r15s`
  - `r1m`
  - `r15m`
  - `ut`
  - `pg`
  - `io`
  - `tmp`
  - `swp`
  - `mem load`

  The `lshosts` and `lsload` commands display “`-`” for each of these items.

- LSF integrated with SLURM only runs daemons on one node within the HP XC system. This node hosts an HP XC LSF Alias, which is an IP address and corresponding **host name** specifically established for LSF integrated with SLURM on HP XC to use. The HP XC system is known by this HP XC LSF Alias within LSF.

  Various LSF commands, such as `lsid`, `lshosts`, and `bhosts`, display HP XC LSF Alias in their output. The default value of the HP XC LSF Alias, `lsfhost.localdomain` is shown in the following examples:

  ```
  $ lsid
  Platform LSF HPC version, Update n, build date stamp
  Copyright 1992-2008 Platform Computing Corporation

  My cluster name is hptclsf
  My master name is lsfhost.localdomain

  $ lshosts
  HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
  lsfhost.loc SLINUX6 Opteron8  60.0     8  2007M      -    Yes (slurm)

  $ bhosts
  HOST_NAME          STATUS   JL/U    MAX  NJOBS    RUN  SSUSP  USUSP  RSV
  lsfhost.localdomai ok          -      8      0      0      0      0  0
  ```

- All HP XC nodes are dynamically configured as “LSF Floating Client Hosts” so that you can execute LSF commands from any HP XC node. When you do execute an LSF command from an HP XC node, an entry in the output of the `lshosts` acknowledges the node is licensed to run LSF commands.

  In the following example, node `n15` is configured as an LSF Client Host, not the LSF execution host. This is shown in the output when you run `lshosts` command is run on that node: The values for the `type` and `model` are `UNKNOWN` and the value for `server` is `No`.

  ```
  $ lshosts
  HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
  lsfhost.loc SLINUX6 Opteron8  60.0     8  2007M      -    Yes (slurm)
  $ ssh n15 lshosts
  HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
  lsfhost.loc SLINUX6 Opteron8  60.0     8  2007M      -    Yes (slurm)
  n15         UNKNOWN UNKNOWN_   1.0     -      -      -     No ()
  ```

- The job-level run-time limits enforced by LSF integrated with SLURM are not supported.
- LSF integrated with SLURM does not support parallel or SLURM-based interactive jobs in PTY mode (`bsub -Is` and `bsub -Ip`). However, after LSF dispatches a user job on the HP XC system, you can use the `srun` or `ssh` command to access the job resources directly accessible. For more information, see “Working Interactively Within an Allocation”.
- LSF integrated with SLURM does not support user-account mapping and system-account mapping.
- LSF integrated with SLURM does not support chunk jobs. If a job is submitted to chunk queue, SLURM lets the job pend.
- LSF integrated with SLURM does not support topology-aware advanced reservation scheduling.

## 10.4 Job Terminology

The following terms are used to describe jobs submitted to LSF integrated with SLURM:

| | |
|---|---|
| Batch job | A job submitted to LSF or SLURM that runs without any I/O connection back to the terminal from which the job was submitted. This job may run immediately, or it may run |

sometime in the future, depending on resource availability and batch system scheduling policies.

Batch job submissions typically provide instructions on I/O management, such as files from which to read input and filenames to collect output.

By default, LSF jobs are batch jobs. The output is e-mailed to the user, which requires that e-mail be set up properly. SLURM batch jobs are submitted with the `srun -b` command. By default, the output is written to `$CWD/slurm-`*`SLURMjobID`*`.out` from the node on which the batch job was launched.

Use Ctrl-C at any time to terminate the job.

| | |
|---|---|
| Interactive batch job | A job submitted to LSF or SLURM that maintains I/O connections with the terminal from which the job was submitted. The job is also subject to resource availability and scheduling policies, so it may pause before starting. After running, the job output displays on the terminal and the user can provide input if the job allows it. By default, SLURM jobs are interactive. Interactive LSF jobs are submitted with the `bsub -I` command. 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 these constraints: • `mem` • `tmp` • `nodes=1` • `mincpus > 1`. Pseudo-parallel jobs are allocated one node for their exclusive use. |

**NOTE:** 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.

LSF considers this job type as a parallel job because the job requests explicit node resources. LSF does not monitor these additional resources, so it cannot schedule any other jobs to the node without risking resource contention. Therefore LSF

| | |
|---|---|
| | allocates the appropriate whole node for exclusive use by the serial job in the same manner as it does for parallel jobs, hence the name “pseudo-parallel”. |
| 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 the following specifications:<br>• `SLURM[`*`nodes=min-max`*`]` (if specified)<br>• `SLURM[`*`nodelist=node_list`*`]` (if specified)<br>• `bsub -n`<br>Parallel jobs and serial jobs cannot run on the same node. |
| Small job | A parallel job that can potentially fit into a single node, and does not explicitly request more than one node (SLURM[*nodes*] or SLURM[*node_list*] specification). LSF tries to allocate a single node for a small job. |

## 10.5 Using LSF Integrated with SLURM in the HP XC Environment

This section provides some additional information that should be noted about using LSF in the HP XC Environment.

### 10.5.1 Useful Commands

The following describe useful commands for LSF Integrated with SLURM:

- Use the `bjobs -l` and `bhist -l` commands to see the components of the actual SLURM allocation command.
- Use the `bkill` command to kill jobs.
- Use the `bjobs` command to monitor job status in LSF integrated with SLURM.
- Use the `bqueues` command to list the configured job queues in LSF integrated with SLURM.

### 10.5.2 Job Startup and Job Control

When LSF starts a SLURM job, it sets `SLURM_JOBID` to associate the job with the SLURM allocation. While a job is running, all LSF supported operating-system-enforced resource limits are supported, including core limit, CPU time limit, data limit, file size limit, memory limit, and stack limit. If the user kills a job, LSF propagates signals to entire job, including the job file running on the local node and all tasks running on remote nodes.

### 10.5.3 Preemption

LSF uses the SLURM "node share" feature to facilitate preemption. When a low-priority is job preempted, job processes are suspended on allocated nodes, and LSF places the high-priority job on the same node. After the high-priority job completes, LSF resumes suspended low-priority jobs.

## 10.6 Submitting Jobs

The `bsub` command submits jobs to LSF; it is used to request a set of resources on which to launch a job. This section focuses on enhancements to this command from the LSF integration with SLURM on the HP XC system; this section does not discuss standard `bsub` functionality or flexibility. See the Platform LSF documentation and the *bsub*(1) manpage for more information on this important command. The topic of submitting jobs with the LSF-SLURM External Scheduler is explored in detail in “Submitting a Parallel Job Using the SLURM External Scheduler”.

The HP XC system has several features that make it optimal for running parallel applications, particularly (but not exclusively) **MPI** applications. You can use the `bsub` command's `-n` to request more than one core for a job. This option, coupled with the external SLURM scheduler, discussed in "LSF-SLURM External Scheduler", gives you much flexibility in selecting resources and shaping how the job is executed on those resources.

LSF reserves the requested number of nodes and executes one instance of the job on the first reserved node, when you request multiple nodes. Use the `srun` command or the `mpirun` command with the `-srun` option in your jobs to launch parallel applications. The `-srun` can be set implicitly for the `mpirun` command; see "Submitting a Parallel Job That Uses the HP-MPI Message Passing Interface" for more information on using the `mpirun -srun` command.

Most parallel applications rely on `rsh` or `ssh` to "launch" remote tasks. The `ssh` utility is installed on the HP XC system by default. If you configured the **ssh** keys to allow unprompted access to other nodes in the HP XC system, the parallel applications can use `ssh`. See "Enabling Remote Execution with OpenSSH" for more information on `ssh`.

The following table shows exit codes for jobs launched under LSF integrated with SLURM:

**Table 10-1 LSF with SLURM Job Launch Exit Codes**

| Exit Code | Description |
|---|---|
| 0 | Success |
| 124 | There was a job launch error in SLURM |
| 125 | There was a job launch error in HPC-LSF |

## 10.7 LSF-SLURM External Scheduler

The external scheduler option is an important option that can be included when submitting parallel jobs with LSF integrated with SLURM. This option

- Provides application-specific external scheduling options for jobs capabilities
- Lets you include several SLURM options in the LSF command line.

For example, you can submit a job to run one task per node when you have a resource-intensive job that needs to have sole access to the node's full resources. If your job needs particular resources found only on a specific set of nodes, you can use this option to submit a job to those nodes.

The LSF host options enable you to identify an HP XC system "host" within a larger LSF cluster. After the HP XC system is selected, LSF's external SLURM scheduler provides the additional flexibility to request specific resources within the HP XC system

You can use the LSF external scheduler functionality within the `bsub` command and in LSF queue configurations. See the LSF *bqueues*(1) command for more information on determining how the available queues are configured on HP XC systems.

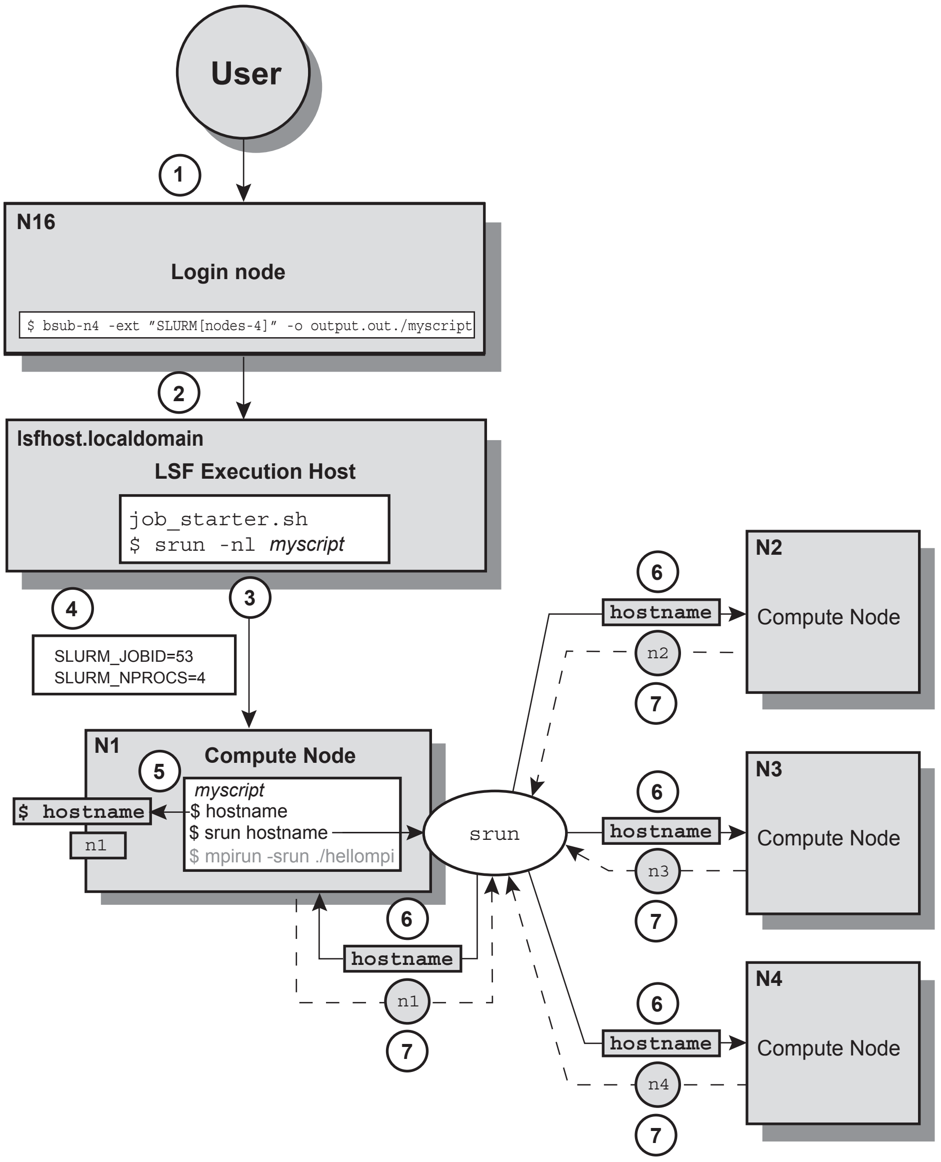
See "Submitting a Parallel Job Using the SLURM External Scheduler" for information and examples on submitting jobs with the LSF-SLURM External Scheduler.

## 10.8 How LSF and SLURM Launch and Manage a Job

This section describes what happens in the HP XC system when a job is submitted to LSF. Figure 10-1 illustrates this process. Use the numbered steps in the text and depicted in the illustration as an aid to understanding the process.

Consider the HP XC system configuration shown in Figure 10-1, in which `lsfhost.localdomain` is the virtual IP name assigned to the **LSF execution host**, node `n16` is the login node, and nodes `n[1-10]` are compute nodes in the `lsf` partition. All nodes contain two cores, providing 20 cores for use by LSF jobs.

**Figure 10-1 How LSF and SLURM Launch and Manage a Job**



1. A user logs in to login node `n16`.
2. The user executes the following LSF `bsub` command on login node `n16`:

   ```
   $ bsub -n4 -ext "SLURM[nodes=4]" -o output.out ./myscript
   ```

   This `bsub` command launches a request for four cores (from the `-n4` option of the `bsub` command) across four nodes (from the `-ext "SLURM[nodes=4]"` option); the job is launched on those cores. The script, `myscript`, which is shown here, runs the job:

   ```
   #!/bin/sh
   hostname
   srun hostname
   mpirun -srun ./hellompi
   ```

3. LSF schedules the job and monitors the state of the resources (**compute nodes**) in the SLURM `lsf` partition. When the LSF scheduler determines that the required resources are available, LSF allocates those resources in SLURM and obtains a SLURM job identifier (`jobID`) that corresponds to the allocation.

   In this example, four cores spread over four nodes (`n1`,`n2`,`n3`,`n4`) are allocated for `myscript`, and the SLURM job id of 53 is assigned to the allocation.

4. LSF prepares the user environment for the job on the LSF execution host node and dispatches the job with the `job_starter.sh` script. This user environment includes standard LSF environment variables and two SLURM-specific environment variables: `SLURM_JOBID` and `SLURM_NPROCS`.

   `SLURM_JOBID` is the SLURM job ID of the job. Note that this is not the same as the LSF `jobID`. "Translating SLURM and LSF JOBIDs" describes the relationship between the `SLURM_JOBID` and the LSF JOBID.

   `SLURM_NPROCS` is the number of processes allocated.

   These environment variables are intended for use by the user's job, whether it is explicitly (user scripts may use these variables as necessary) or implicitly (the `srun` commands in the user's job use these variables to determine its allocation of resources).

   The value for `SLURM_NPROCS` is 4 and the `SLURM_JOBID` is 53 in this example.

5. The user job `myscript` begins execution on compute node `n1`.

   The first line in `myscript` is the `hostname` command. It executes locally and returns the name of node, `n1`.

6. The second line in the `myscript` script is the `srun hostname` command. The `srun` command in `myscript` inherits `SLURM_JOBID` and `SLURM_NPROCS` from the environment and executes the `hostname` command on each compute node in the allocation.
7. The output of the `hostname` tasks (`n1`, `n2`, `n3`, and `n4`). is aggregated back to the `srun` launch command (shown as dashed lines in Figure 10-1), and is ultimately returned to the `srun` command in the job starter script, where it is collected by LSF.

The last line in `myscript` is the `mpirun -srun ./hellompi` command. The `srun` command inside the `mpirun` command in `myscript` inherits the `SLURM_JOBID` and `SLURM_NPROCS` environment variables from the environment and executes `hellompi` on each compute node in the allocation.

The output of the `hellompi` tasks is aggregated back to the `srun` launch command where it is collected by LSF.

The command executes on the allocated compute nodes `n1`, `n2`, `n3`, and `n4`.

When the job finishes, LSF cancels the SLURM allocation, which frees the compute nodes for use by another job.

## 10.9 Determining the LSF Execution Host

The `lsid` command displays the name of the HP XC system, the name of the LSF execution host, and some general LSF information.

```
$ lsid
Platform LSF HPC version, Update n, build date stamp
Copyright 1992-2008 Platform Computing Corporation

My cluster name is hptclsf
My master name is lsfhost.localdomain
```

In this example, `hptclsf` is the LSF cluster name (where is user is logged in and which contains the compute nodes), and `lsfhost.localdomain` is the virtual IP name of the node where LSF is installed and runs (LSF execution host).

## 10.10 Determining Available System Resources

For the best use of system resources when launching an application, it is useful to know the system resources that are available for your use. This section describes how to obtain information about system resources such as the number of cores available, LSF execution host node information, and LSF system queues.

## 10.10.1 Examining System Core Status

The `bhosts` command displays LSF resource usage information. This command is useful to examine the status of the system cores. The `bhosts` command provides a summary of the jobs on the system and information about the current state of LSF. For example, it can be used to determine if LSF is ready to start accepting batch jobs.

LSF daemons run on only one node in the HP XC system, so the `bhosts` command will list one host, which represents all the resources of the HP XC system. The total number of cores for that host should be equal to the total number of cores assigned to the SLURM `lsf` partition.

By default, this command returns the host name, host status, and job state statistics.

The following example shows the output from the `bhosts` command:

```
$ bhosts
HOST_NAME           STATUS JL/U MAX  NJOBS RUN SSUSP USUSP RSV
lsfhost.localdomain   ok      -  16    0     0     0     0    0
```

Of note in the `bhosts` output:

- The `HOST_NAME` column displays the name of the LSF execution host.
- The `MAX` column displays the total core count (usable cores) of all available computer nodes in the `lsf` partition.
- The `STATUS` column shows the state of LSF and displays a status of either `ok` or `closed`.
- The `NJOBS` column displays the number of jobs. Note that in LSF terminology, a parallel job with 10 tasks counts as 10 jobs.

## 10.10.2 Getting Information About the LSF Execution Host Node

The `lshosts` command displays resource information about the LSF cluster. This command is useful for verifying machine-specific information.

LSF daemons run on only one node in the HP XC system, so the `lshosts` command will list one host — which represents all the resources assigned to it by the HP XC system. The total number of cores for that host should be equal to the total number of cores assigned to the SLURM `lsf` partition.

By default, `lshosts` returns the following information: host name, host type, host model, core factor, number of cores, total memory, total swap space, server information, and static resources.

The following example shows the output from the `lshosts` command:

```
$ lshosts
HOST_NAME   type    model    cpuf ncpus maxmem maxswp server RESOURCES
lsfhost.loc SLINUX6 Itanium2 16.0    12  3456M      -    Yes (slurm)
n7          UNKNOWN UNKNOWN_  1.0     -      -      -     No ()
n8          UNKNOWN UNKNOWN_  1.0     -      -      -     No ()
n2          UNKNOWN UNKNOWN_  1.0     -      -      -     No ()
```

Of note in the `lshosts` output:

- The `HOST_NAME` column displays the name of the LSF execution host, `lsfhost.localdomain` and any other HP XC nodes that have been granted a floating client license because LSF commands were executed on them. LSF does not know about these floating client hosts, so they are listed as `UNKNOWN` types and models.
- The `type` column displays the type of resource. This value is `SLINUX64` for all HP XC systems.
- The `ncpus` column displays the total core count (usable cores) of all available computer nodes in the `lsf` partition.
- The `maxmem` column displays minimum `maxmem` over all available computer nodes in the `lsf` partition.
- The `maxtmp` column (not shown) displays minimum `maxtmp` over all available computer nodes in the `lsf` partition. Use the `lshosts -l` command to display this column.

## 10.10.3 Getting Host Load Information

The LSF `lsload` command displays load information for LSF execution hosts.

```
$ lsload
HOST_NAME    status  r15s  r1m  r15m  ut  pg  ls  it  tmp  swp  mem
lsfhost.loc   ok        -    -     -   -   -   4   -    -    -    -
```

In the previous example output, the LSF execution host (`lsfhost.localdomain`) is listed under the `HOST_NAME` column. The `status` is listed as `ok`, indicating that it can accept remote jobs. The `ls` column shows the number of current login users on this host.

See the `OUTPUT` section of the `lsload` manpage for further information about the output of this example. In addition, see the Platform LSF documentation and the *lsload*(1) manpage for more information about the features of this command.

For individual compute node load information, see the discussion on `metrics` in *shownode*(1).

## 10.10.4 Examining System Queues

All jobs on the HP XC system that are submitted to LSF are placed into an LSF job queue. HP recommends that you examine the status and availability of LSF system queues before launching a job so that you can select the most appropriate queue for your job.

You can easily verify the status, limits, and configurations of LSF queues with the `bqueues` command. This command is fully described in Platform LSF documentation and manpages.

See *bsub*(1) for more information on submitting jobs to specific queues.

For more information on the `bqueues` command, see *bqueues*(1).

## 10.10.5 Getting Information About the `lsf` Partition

Information about the SLURM `lsf` compute node partition can be viewed with the SLURM `sinfo` command. A partition is one or more compute nodes that have been grouped together. A SLURM `lsf` partition is created when the HP XC system is installed. This partition contains the resources that will be managed by LSF and available for jobs submitted to LSF.

The `sinfo` command reports the state of the `lsf` partition and all other partitions on the system. The `sinfo` command displays a summary of available partition and node information such as partition names, nodes/partition, and cores/node). It has a wide variety of filtering, sorting, and formatting options.

The following example shows the use of the `sinfo` command to obtain `lsf` partition information:

```
$ sinfo -p lsf
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite   128   idle n[1-128]
```

Use the following command to obtain more information on the nodes in the `lsf` partition:

```
$ sinfo -p lsf -lNe
date and time
NODELIST    NODES PARTITION STATE CPUS   S:C:T MEMORY TMP_DISK WEIGHT FEATURES REASON
n[1-128]        3       lsf  idle    2   2:1:1   3892        0      1   (null) none
```

See "Getting System Information with the sinfo Command" and the *sinfo*(1) manpage and for further information about using the `sinfo` command.

## 10.11 Getting Information About Jobs

There are several ways you can get information about a specific job after it has been submitted to LSF integrated with SLURM. This section briefly describes some of the commands that are available under LSF integrated with SLURM to gather information about a job. This section is only intended to give you an idea of the commonly used commands and to describe any differences there may be in their operation in the HP XC environment, not as a complete reference on this topic. See the LSF manpages for full information about the commands described in this section.

The following LSF commands are described in this section:

| | |
|---|---|
| `bjobs` | "Examining the Status of a Job" |
| `bhist` | "Viewing the Historical Information for a Job" |

### 10.11.1 Getting Job Allocation Information

Before a job runs, LSF integrated with SLURM allocates SLURM compute nodes based on job resource requirements.

After LSF integrated with SLURM allocates nodes for a job, it attaches allocation information to the job.

The `bjobs -l` command provides job allocation information on running jobs. The `bhist -l` command provides job allocation information for a finished job. For details about using these commands, see the LSF manpages .

A job allocation information string resembles the following:

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

This allocation string has the following values:

| | |
|---|---|
| `slurm_id` | `SLURM_JOBID` environment variable. This is SLURM allocation ID (Associates LSF job with SLURM allocated resources.) |
| `ncpus` | `SLURM_NPROCS` environment variable. This the actual number of allocated cores. Under node-level allocation scheduling, this number may be bigger than what the job requests.) |
| `slurm_alloc` | A comma separated list of allocated nodes. |

LSF integrated with SLURM sets the `SLURM_JOBID` and `SLURM_NPROCS` environment variables, when it starts a job.

Example 10-3 illustrates how to use the the `bjobs -l` command to obtain job allocation information about a running job:

**Example 10-3 Job Allocation Information for a Running Job**

```
$ bjobs -l 24
Job <24>, User <lsfadmin>, Project <default>,
                     Status <RUN>, Queue <normal>,
                     Interactive pseudo-terminal shell mode,
                     Extsched <SLURM[nodes=4]>, Command </bin/bash>

date and time stamp: Submitted from host <n2>, CWD <$HOME>,
                     4 Processors Requested, Requested Resources <type=any>;
date and time stamp: Started on 4 Hosts/Processors <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];

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        lsfadmin   date and time stamp  SLURM[nodes=4]    N
```

In particular, note the node and job allocation information provided in the above output:

```
date and time stamp: Started on 4 Hosts/Processors <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];
```

Example 10-4 illustrates how to use the output obtained using the `bhist -l` command to obtain job allocation information about a job that has run:

**Example 10-4 Job Allocation Information for a Finished Job**

```
$ bhist -l 24
Job <24>, User <lsfadmin>, Project <default>,
                     Interactive pseudo-terminal shell mode,
                     Extsched <SLURM[nodes=4]>, Command </bin/bash>

date and time stamp: Submitted from host <n2>, to
                     Queue <normal>, CWD <$HOME>,
                     4 Processors Requested, Requested Resources <type=any>;

date and time stamp: Dispatched to 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];date and time stamp: Starting (Pid 4785);
date and time stamp: Done successfully.
The CPU time used is 0.1 seconds;
date and time stamp: Post job process done successfully;

Summary of time in seconds spent in various states by date and time stamp
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN   TOTAL
  11       0        220      0        0        0       231
```

In particular, note the node and job allocation information provided in the above output:

```
date and time stamp: Dispatched to 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];
```

## 10.11.2 Examining the Status of a Job

Once a job is submitted, you can use the `bjobs` command to track the job's progress. The `bjobs` command reports the status of a job submitted to LSF. By default, `bjobs` lists only the user's jobs that have not finished or exited.

The following are examples of `bjobs` command usage, and show the output it produces on an HP XC system. For more information about the `bjobs` command and its output, see the LSF manpages.

Example 10-5 provides abbreviated output of the `bjobs` command.

**Example 10-5 Using the bjobs Command (Short Output)**

```
$ bjobs 24
JOBID USER   STAT QUEUE  FROM_HOST EXEC_HOST           JOB_NAME  SUBMIT_TIME
24    msmith RUN  normal n16       lsfhost.localdomain /bin/bash date and time
```

As shown in the previous output, the `bjobs` command returns information that includes the job id, user name, job status, queue name, submitting host, executing host, job name, and submit time. In this example, the output shows that job `/bin/bash` was submitted from node `n16` and launched on the execution host (`lsfhost.localdomain`).

Example 10-6 provides extended output of the `bjobs` command.

**Example 10-6 Using the bjobs Command (Long Output)**

```
$ bjobs -l 24
Job <24>, User <msmith>,Project <default>,Status <RUN>,
                     Queue <normal>, Interactive pseudo-terminal shell
                     mode, Extsched <SLURM[nodes=4]>, Command </bin/bash>
date and time stamp: Submitted from host <n16>, CWD <$HOME>,
                     4 Processors Requested, Requested Resources <type=any>;

date and time stamp: Started on 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];

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        lsfadmin   date and time stamp  SLURM[nodes=4]    N
```

## 10.11.3 Viewing the Historical Information for a Job

The LSF `bhist` command is a good tool for tracking the lifetime of a job within LSF. The `bhist` command provides detailed information about a job, including running, pending, and suspended jobs, information such as the amount of time spent in various states, and in-depth information about a job's progress while the job was under LSF control.

See the LSF `bhist` manpage for more information about this command, its options, and its output.

A brief summary about a finished job can be obtained with the `bhist` command, shown in Example 10-7. This command provides statistics about the amount of time that the job has spent in various states.

**Example 10-7 Using the bhist Command (Short Output)**

```
$ bhist 24
Summary of time in seconds spent in various states:
JOBID  USER   JOB_NAME  PEND   PSUSP RUN   USUSP SSUSP UNKWN TOTAL
24     smith  bin/bash  11      0    220    0     0     0     231
```

The information in the output provided by this example is explained in Table 10-2.

**Table 10-2 Output Provided by the bhist Command**

| Field | Description |
|---|---|
| `JOBID` | The job ID that LSF assigned to the job. |
| `USER` | The user who submitted the job. |
| `JOB_NAME` | The job name assigned by the user. |
| `PEND` | The total waiting time, excluding user suspended time, before the job is dispatched. |
| `PSUSP` | The total user suspended time of a pending job. |
| `RUN` | The total run time of the job. |
| `USUSP` | The total user suspended time after the job is dispatched. |
| `SSUSP` | The total system suspended time after the job is dispatched. |

**Table 10-2 Output Provided by the bhist Command** *(continued)*

| Field | Description |
|---|---|
| UNKWN | The total unknown time of the job. |
| TOTAL | The total time that the job has spent in all states. |

For detailed information about a finished job, add the `-l` option to the `bhist` command, shown in Example 10-8. The `-l` option specifies that the long format is requested.

**Example 10-8 Using the bhist Command (Long Output)**

```
$ bhist -l 24
Job <24>, User <lsfadmin>, Project <default>,
Interactive pseudo-terminal shell mode,
Extsched <SLURM[nodes=4]>, Command </bin/bash>
date and time stamp: Submitted from host <n2>,
to Queue <normal>, CWD <$HOME>,
4 Processors Requested, Requested Resources <type=any>;

date and time stamp: Dispatched to 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];
date and time stamp: Starting (Pid 4785);

Summary of time in seconds spent in various states by
date and time stamp
  PEND     PSUSP    RUN     USUSP    SSUSP    UNKWN    TOTAL
  11       0        124     0        0        0        135
```

## 10.12 Translating SLURM and LSF JOBIDs

LSF and SLURM are independent resource management components of the HP XC system. They maintain their own job identifiers (JOBIDs). It may be useful to be able to determine which the `SLURM_JOBID` environment variable matches an LSF JOBID, and vice versa.

When a job is submitted to LSF, it is given an LSF JOBID, as in this example:

```
$ bsub -o %J.out -n 8 sleep 300
   Job <99> is submitted to default queue <normal>
```

The following is the sequence of events when a SLURM JOBID is assigned:

- No `SLURM_JOBID` exists while the job is PENDing in LSF.
- After LSF determines that the resources are available in SLURM for this job, LSF requests an allocation in SLURM.
- After the SLURM allocation is established, there is a corresponding SLURM JOBID for the LSF JOBID.

Use the `bjobs` command to view the SLURM JOBID:

```
$ bjobs -l 99 | grep slurm
date and time stamp: slurm_id=123;ncpus=8;slurm_alloc=n[13-16];
```

The SLURM JOBID is `123` for the LSF JOBID `99`.

You can also find the allocation information in the output of the `bhist` command:

```
$ bhist -l 99 | grep slurm
date and time stamp: slurm_id=123;ncpus=8;slurm_alloc=n[13-16];
```

When LSF creates an allocation in SLURM, it constructs a name for the allocation by combining the LSF cluster name with the LSF JOBID. You can see this name with the `scontrol` and `sacct` commands while the job is running:

```
$ scontrol show job | grep Name
  Name=hptclsf@99
```

```
$ sacct -j 123
Jobstep    Jobname            Partition    Ncpus Status     Error
---------- ------------------ ---------- ------- ---------- -----
123        hptclsf@99         lsf              8 RUNNING    0
123.0      hptclsf@99         lsf              0 RUNNING    0
```

In these examples, the job name is `hptclsf@99`; the LSF job ID is `99`.

Note that the `scontrol show job` command keeps jobs briefly after they finish, then it purges itself; this is similar with the `bjobs` command. The `sacct` command continues to provide job information after the job has finished; this is similar to `bhist` command:

```
$ sacct -j 123
Jobstep    Jobname            Partition    Ncpus Status     Error
---------- ------------------ ---------- ------- ---------- -----
123        hptclsf@99         lsf              8 CANCELLED  0
123.0      hptclsf@99         lsf              0 COMPLETED  0
```

The status of a completed job handled by LSF is always `CANCELLED` because LSF destroys the allocation that it creates for the job after the user job completes. LSF performs the following steps:

- Creates the allocation in SLURM.
- Submits the user job to SLURM.
- Waits for the user job to finish.
- Cancels the allocation in SLURM.

The example shown above had three entries for a completed job. There are at least two entries; the number of entries depends on the construction of the user job:

- The first entry represents the allocation created by LSF.
- The second entry, SLURM job step 0, represents the user job that LSF submits to SLURM.
- Further entries represent `srun` or `mpirun -srun` commands invoked by the user job.

## 10.13 Working Interactively Within an Allocation

The best way to work interactively on HP XC is to separate the allocation from the interactive work. In one terminal, submit your allocation request to LSF with an interactive shell as the job. For example:

```
$ bsub -I -n4 -ext "SLURM[nodes=4]" /bin/bash
Job <124> is submitted to the default queue <interactive>.
<<Waiting for dispatch...>>
<<Starting on lsfhost.localdomain>>
```

The `bsub` command requests 4 nodes and runs `/bin/bash` on the first allocated node. If resources are not immediately available, the terminal pauses at `<<Waiting for dispatch...>>`. When `<<Starting on` displays, the resources are allocated and `/bin/bash` is running.

To gather information about this allocation, run the following command in this first terminal (note there is no prompt from the `/bin/bash` process):

```
$ bjobs -l 124 | grep slurm
date and time stamp: slurm_id=150;ncpus=8;slurm_alloc=n[1-4];
```

LSF allocated nodes `n[1-4]` for this job. The SLURM JOBID is 150 for this allocation.

Begin your work in another terminal. Use `ssh` to login to one of the compute nodes. If you want to run tasks in parallel, use the `srun` command with the `--jobid` option to specify the SLURM JOBID. For example, to run the `hostname` command on all nodes in the allocation:

```
$ srun --jobid=150 hostname
n1
n2
n3
n4
```

You can simplify this by first setting the `SLURM_JOBID` environment variable to the SLURM JOBID in the environment, as follows:

```
$ export SLURM_JOBID=150
$ srun hostname
n1
n2
n3
n4
```

**Note:**

Be sure to unset the `SLURM_JOBID` when you are finished with the allocation, to prevent a previous SLURM JOBID from interfering with future jobs:

```
$ unset SLURM_JOBID
```

The following examples illustrate launching interactive MPI jobs. They use the `hellompi` job script introduced in Section 5.3.2 (page 52). Example 10-9

### Example 10-9 Launching an Interactive MPI Job

```
$ mpirun -srun --jobid=150 hellompi
Hello! I'm rank 0 of 4 on n1
Hello! I'm rank 1 of 4 on n2
Hello! I'm rank 2 of 4 on n3
Hello! I'm rank 3 of 4 on n4
```

Example 10-10 uses the `-n 8` option to launch on all cores in the allocation.

### Example 10-10 Launching an Interactive MPI Job on All Cores in the Allocation

This example assumes 2 cores per node.

```
$ mpirun -srun --jobid=150 -n8 hellompi
Hello! I'm rank 0 of 8 on n1
Hello! I'm rank 1 of 8 on n1
Hello! I'm rank 2 of 8 on n2
Hello! I'm rank 3 of 8 on n2
Hello! I'm rank 4 of 8 on n3
Hello! I'm rank 5 of 8 on n3
Hello! I'm rank 6 of 8 on n4
Hello! I'm rank 7 of 8 on n4
```

Alternatively, you can use the following:

```
$ export SLURM_JOBID=150
$ export SLURM_NPROCS=8
$ mpirun -srun hellompi
Hello! I'm rank 0 of 8 on n1
Hello! I'm rank 1 of 8 on n1
Hello! I'm rank 2 of 8 on n2
Hello! I'm rank 3 of 8 on n2
Hello! I'm rank 4 of 8 on n3
Hello! I'm rank 5 of 8 on n3
Hello! I'm rank 6 of 8 on n4
Hello! I'm rank 7 of 8 on n4
```

Use **ssh** to launch a Totalview debugger session, assuming that TotalView is installed and licensed and that `ssh X forwarding` is properly configured:

```
$ export SLURM_JOBID=150
$ export SLURM_NPROCS=4
$ mpirun -tv srun additional parameters as needed
```

After you finish with this interactive allocation, exit the `/bin/bash` process in the first terminal; this ends the LSF job.

**Note:**

If you exported any variables, such as `SLURM_JOBID` and `SLURM_NPROCS`, be sure to `unset` them as follows before submitting any further jobs from the second terminal:

```
$ unset SLURM_JOBID
$ unset SLURM_NPROCS
```

You do not need to launch the `/bin/bash` shell to be able to interact with any compute node resources; any running job will suffice. This is excellent for checking on long-running jobs. For example, if you had submitted a CPU-intensive job, you could execute the `uptime` command on all nodes in the allocation to confirm an expected high load on the nodes. The following is an example of this; the LSF JOBID is 200 and the SLURM JOBID is 250:

```
$ srun --jobid=250 uptime
```

If you are concerned about allocating the resources too long or leaving them allocated long after you finished using them, you could submit a simple `sleep` job to limit the allocation time, as follows:

```
$ bsub -n4 -ext "SLURM[nodes=4]" -o %J.out sleep 300
Job <125> is submitted to the default queue <normal>.
```

After you confirm that this job is running using the `bjobs -l 125` command, you can operate interactively for with the resources. If LSF observes no SLURM activity within the allocation after 5 minutes, it terminates the job. Any existing SLURM activity (including running MPI jobs with the `mpirun -srun` command) is allowed to continue.

## 10.14 LSF Equivalents of SLURM srun Options

Table 10-3 describes the `srun` options and lists their LSF equivalents.

**Table 10-3 LSF Equivalents of SLURM srun Options**

| `srun` Option | Description | LSF Equivalent |
|---|---|---|
| `-n`<br>`--ntasks=`*`ntasks`* | Number of processes (tasks) to run. | `bsub -n` *`num`* |
| `-c`<br>`--processors-per-task=`*`ntasks`* | Specifies the number of cores per task. Min processors per node = MAX(ncpus, mincpus) | HP XC does not provide this option because the meaning of this option can be covered by `bsub -n` and `mincpus=`*`n`*. |
| `-N`<br>`--nodes=`*`min`*`[-`*`max`*`]` | Specifies the minimum and, optionally, maximum number of nodes allocated to job. The job allocation will contain at least the minimum number of nodes. | `-ext "SLURM[nodes=`*`minmax`*`]"`<br>where *`minmax`* is *`min`*`[-`*`max`*`]` |
| `--mincpus=`*`n`* | Specifies the minimum number of cores per node. Min processors per node = MAX(-c ncpus, --mincpus=n). Default value is 1. | `-ext "SLURM[mincpus=`*`n`*`]"` |
| `--mem=`*`MB`* | Specifies a minimum amount of real memory of each node. By default, job does not require `-ext`. | `-ext "SLURM[mem=`*`MB`*`]"` |
| `--tmp=`*`MB`* | Specifies a minimum amount of temporary disk space of each node. By default, job does not require `-ext`. | `-ext "SLURM[tmp=`*`MB`*`]"` |
| `-C`<br>`--constraint=`*`list`* | Specifies a list of constraints. The list may include multiple features separated by the `&` character (meaning ANDed) or the `\|` (meaning ORed). By default, job does not require `-ext`. | `-ext "SLURM[constraint=`*`list`*`]"` |

**Table 10-3 LSF Equivalents of SLURM srun Options** *(continued)*

| `srun` Option | Description | LSF Equivalent |
|---|---|---|
| `-w`<br>`--nodelist=`*`node1,...nodeN`* | Requests a specific list of nodes. The job will at least contain these nodes. The list may be specified as a comma-separated list of nodes, or a range of nodes. By default, job does not require `-ext`. | `-ext`<br>`"SLURM[nodelist=`*`node1,...nodeN`*`]"` |
| `-x`<br>`--exclude=`*`node1,...nodeN`* | Requests that a specific list of hosts be excluded in the resource allocated to this job. By default, job does not require. | `-ext`<br>`"SLURM[exclude=`*`node1,...nodeN`*`]"` |
| `-p`<br>`--partition=`*`partition`* | Requests resources from partition *`partition`*. | You cannot use this option. The `lsf` partition is the only one provided. |
| `--contiguous` | Requests a contiguous range of nodes. By default, job does not require contiguous nodes. | `-ext "SLURM[contiguous=yes]"` |
| `-o`<br>`--output=`*`file_name`* | Specifies the mode for `stdout` redirection. | `bsub -o` *`output_file`* |
| `-o`<br>`--output=none,tasked` | Specify the mode for `stdout` redirection. | You can use this option when launching parallel tasks. |
| `-i`<br>`--input=`*`file_name`* | Specify how `stdin` is to be redirected. | `bsub -i` *`input_file`* |
| `-i`<br>`--input=none,tasked` | Specify how `stdin` is to be redirected. | You can use when launching parallel tasks. |
| `-e`<br>`--error=`*`file_name`* | Specify how `stderr` is to be redirected. | `bsub -e` *`error_file`* |
| `-e`<br>`--error=none,tasked` | Specify how `stderr` is to be redirected. | You can use when launching parallel tasks. |
| `-J`<br>`--job-name=`*`job_name`* | Specify a name for the job. | You cannot use this option. When creating allocation. SLURM sets LSF job id automatically. |
| `--uid=`*`user`* | Root attempts to submit or run a job as normal user. | You cannot use this option. LSF uses it to create allocation. |
| `-t`<br>`--time=`*`minutes`* | Establish a time limit to terminate the job after specified number of minutes. | `bsub -W` *`runlimit`* |
| `--gid=`*`group`* | Root attempts to submit or run a job as group. | You cannot use this option. LSF uses this option to create allocation. |
| `-A`<br>`--allocate` | Allocate resource and spawn a shell. | You cannot use this option. LSF uses this option to create allocation. |
| `--no-shell` | Immediately exit after allocating resources. | You cannot use this option. LSF uses this option to create allocation. |
| `-I`<br>`--immediate` | Allocate immediately or fail. | You cannot use this option. LSF uses this option to create allocation. |

**Table 10-3 LSF Equivalents of SLURM srun Options** *(continued)*

| `srun` Option | Description | LSF Equivalent |
|---|---|---|
| `-s`<br>`--share` | Share nodes with other running jobs.<br>`SHARED=FORCE` shares all nodes in partition.<br>`SHARED=YES` shares nodes if and only if `-share` is specified.<br>`SHARED=NO` means do not share the node. | You cannot use this option. LSF uses this option to create allocation. |
| `-O`<br>`--overcommit` | Overcommit resources. | Use when launching parallel tasks. |
| `-b`<br>`--batch` | Submit in "batch mode". | Meaningless under LSF integrated with SLURM. |
| `-r`<br>`--relative=`*`n`* | Run a job step relative to node n of the current allocation. It is about placing tasks within allocation. | Use as an argument to `srun` when launching parallel tasks. |
| `-D`<br>`--chdir=`*`path`* | Specify the working directory of the job. | Job starts in job submission directory by default. |
| `-k`<br>`--no-kill` | Do not automatically terminate a job if one of the nodes it has been allocated fails. | When creating allocation, LSF does not use the `-k` option. SLURM always terminates a job if one of allocated nodes fails |
| `-T`<br>`--threads=`*`nthreads`* | The `srun` command uses *`nthreads`* to initiate and control the parallel job. | Use as an argument to `srun` when launching parallel tasks. |
| `-l`<br>`--label` | Prepend task number to lines of `stdout`/`stderr`. | Use as an argument to `srun` when launching parallel tasks. |
| `-u`<br>`--unbuffered` | Do not line buffer `stdout` from remote tasks. | Use as an argument to `srun` when launching parallel tasks. |
| `-m`<br>`--distribution=`*`block`*\|*`cyl`* | Distribution method for remote processes. | Use as an argument to `srun` when launching parallel tasks. |
| `--mpi=`*`mpi_type`* | Identify the type of **MPI** to be used. | Use as an argument to `srun` when launching parallel tasks. |
| `--jobid=`*`id`* | Initiate a job step under an already allocated job id. | Do not use. `SLURM_JOBID` is set in job environment to associate job with SLURM allocation. |
| `-v` | Verbose operation. | Use as an argument to `srun` when launching parallel tasks. |
| `-d`<br>`--slurm-debug=`*`level`* | A debug level for `slurmd`. | Use as an argument to `srun` when launching parallel tasks. |
| `-W`<br>`--wait=`*`seconds`* | How long to wait after the first task terminates before terminating all remaining tasks. | Use as an argument to `srun` when launching parallel tasks. |
| `-q`<br>`--quit-on-interrupt` | Quit immediately on single `SIGINT`. | Meaningless under LSF integrated with SLURM. |

**Table 10-3 LSF Equivalents of SLURM srun Options** *(continued)*

| `srun` Option | Description | LSF Equivalent |
|---|---|---|
| `-Q`<br>`--quiet` | Suppress informational message. | Use as an argument to `srun` when launching parallel tasks. |
| `--core=`*`type`* | Adjust `corefile` format for parallel job. | Use as an argument to `srun` when launching parallel tasks. |
| `-a`<br>`--attach=`*`id`* | Attach `srun` to a running job. | Meaningless under LSF integrated with SLURM. |
| `-j`<br>`--join` | Join with running job. | Meaningless under LSF integrated with SLURM. |
| `-s`<br>`--steal` | Steal connection to running job. | Meaningless under LSF integrated with SLURM. |

# 11 Advanced Topics

This chapter covers topics intended for the advanced user. This chapter addresses the following topics:

## 11.1 Enabling Remote Execution with OpenSSH

To reduce the risk of network attacks and increase the security of your HP XC system, the traditional `rsh`, `rlogin`, and `telnet` tools are disabled by default, and OpenSSH is provided instead. HP XC systems use the OpenSSH package to provide a more secure environment than the traditional `rsh`, `rlogin`, and `telnet` tools. OpenSSH provides a secure, encrypted connection between your system and the HP XC system.

However, OpenSSH requires, by default, that users enter their password every time that they attempt to access a remote system. When first you use the `ssh` command to access the system, or when attempting to use a tool such as TotalView to debug an application on the **cluster**, you may be prompted to enter your password multiple times. To eliminate the multiple requests, use the procedure described below. Alternatively, you can use the `ssh_create_shared_keys` command, which is described in "Using the Secure Shell to Log In"

There are a number of ways in which to manage an OpenSSH environment. If you have not already established your own procedures, the following procedure will help you to get started. This procedure must be executed by each user, and assumes that your home directory is on a file system that is shared across the cluster.

Log in to your account on the system and execute the following commands. Press the ENTER key in response to all questions.

```
$ ssh-keygen -t dsa
$ cd ~/.ssh
$ cat id_dsa.pub >>authorized_keys
$ chmod go-rwx authorized_keys
```

## 11.2 Running an X Terminal Session from a Remote Node

This section illustrates running an X terminal session from a remote node. An X terminal session can be invoked through SLURM or through LSF. Note that the procedure described in this section creates an unencrypted X terminal session, which is considered unsecure. The information in this section assumes the following:

- You are on a Linux, UNIX, or UNIX-like machine.
- You are on a non-HP XC machine, which is serving your local display.
- Your machine is on a trusted network.

### Step 1. Determining IP Address of Your Local Machine

To begin, you need to determine the IP address of your monitor's display server (the machine serving your monitor), as shown in the following steps. You will use this IP address in later commands to run the X terminal session.

First, echo the display:

```
$ echo $DISPLAY
:0
```

Next, get the name of the local machine serving your display monitor:

```
$ hostname
mymachine
```

Then, use the host name of your local machine to retrieve its IP address:

```
$ host mymachine
mymachine has address 192.0.2.134
```

## Step 2. Logging in to HP XC System

Next, you need to log in to a login node on the HP XC system. For example:

```
$ ssh user@xc-node-name
```

Once logged in to the HP XC system, you can start an X terminal session using SLURM or LSF. Both methods are described in the following sections.

## Step 3. Running an X terminal Session Using SLURM

This section shows how to create an X terminal session on a remote node using SLURM. First, examine the available nodes on the HP XC system. For example:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     2   idle n[46,48]
```

According to the information returned about this HP XC system, SLURM has two nodes, `n46` and `n48`, available for use.

Start an X terminal session on this node, using the information you obtained about your display server to direct output back to it. For example:

```
$ srun -N1 xterm -display 192.0.2.134:0.0
```

The options used in this command are:

| | |
|---|---|
| `srun -N1` | run the job on 1 node |
| xterm | the job is an X terminal session |
| `-display <address>` | monitor's display server address |

Once the job starts, an X terminal session appears on your desktop from the available remote HP XC node. You can verify that the X terminal session is running on a compute node with the `hostname` command. For example:

```
$ hostname
n47
```

You can verify that SLURM has allocated the job as you specified. For example:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     2   idle n[46,48]
$ squeue
JOBID  PARTITION  NAME   USER       ST    TIME  NODES  NODELIST
135      srun     xterm  username    R    0:13    1    n47
```

Exiting from the X terminal session ends the SLURM job.

## Step 4. Running an X terminal Session Using LSF

This section shows how to create an X terminal session on a remote node using LSF. In this example, suppose that you want to use LSF to reserve 4 cores (2 nodes) and start an X terminal session on one of them.

First, examine the available nodes on the HP XC system. For example:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     2   idle n[46,48]
```

According to the information returned about this HP XC system, LSF has two nodes available for use, n46 and n48.

Determine the address of your monitor's display server, as shown at the beginning of "Running an X Terminal Session from a Remote Node". You can start an X terminal session using this address information in a bsub command with the appropriate options. For example:

```
$ bsub -n4 -Ip srun -n1 xterm -display 192.0.2.134:0.0
Job <159> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
```

The options used in this command are:

| | |
|---|---|
| `-n4` | allocate 4 cores |
| `-Ip` | interact with the X terminal session |
| `srun -n1` | run the job on 1 core |
| xterm | the job is an X terminal session |
| `-display <address>` | monitor's display server address |

A remote X terminal session appears on your monitor. The X terminal session job is launched from node n47, which is the LSF execution host node. You can view this job using LSF and SLURM commands. For example:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     2  alloc n[46,48]
$ squeue
JOBID  PARTITION  NAME      USER      ST  TIME   NODES NODELIST
 117       lsf    hptclsf@  username   R  0:25   2     n[46,48]
$ bjobs
JOBID  USER    STAT QUEUE  FROM_HOST EXEC_HOST JOB_NAME   SUBMIT_TIME
119    lsfadmi RUN  norma  n48       4*n47     *8.136:0.0 date and time
```

You can now run some jobs from the X terminal session that you started and make use of the full allocation within the LSF node allocation. For example:

```
$ srun -n4 hostname
n46
n48
n46
n48
```

```
$ srun -n2 hostname
n46
n48
```

Exiting from the X terminal session ends the LSF job.

## 11.3 Using the GNU Parallel Make Capability

By default, the make command invokes the GNU make program. GNU make has the ability to make independent targets concurrently. For example, if building a program requires compiling 10 source files, and the compilations can be done independently, make can manage multiple compilations at once — the number of jobs is user selectable. More precisely, each target's rules are run normally (sequentially within the rule). Typically the rules for an object file target is a single compilation line, so it is common to talk about concurrent compilations, though GNU make is more general.

On non-cluster platforms or command nodes, matching concurrency to the number of cores often works well. It also often works well to specify a few more jobs than cores so that one job can proceed while another is waiting for I/O. On an HP XC system, there is the potential to use compute nodes to do compilations, and there are a variety of ways to make this happen.

One way is to prefix the actual compilation line in the rule with an `srun` command. So, instead of executing `cc foo.c -o foo.o` it would execute `srun cc foo.c -o foo.o`. With concurrency, multiple command nodes would have multiple `srun` commands instead of multiple `cc` commands. For projects that recursively run `make` on subdirectories, the recursive `make` can be run on the compute nodes. For example:

```
$ cd subdir; srun $(MAKE)...
```

Further, if the recursive make is run remotely, it can be told to use concurrency on the remote node. For example:

```
$ cd subdir; srun -n1 -N1 $(MAKE) -j4...
```

This can cause multiple makes to run concurrently, each building their targets concurrently. The `-N1` option is used to reserve the entire node, because it is intended to be used for multiple compilations. The following examples illustrate these ideas. In GNU `make`, a `$(VARIABLE)` that is unspecified is replaced with nothing. Therefore, not specifying `PREFIX` keeps the original Makefile's behavior, but specifying `PREFIX` to appropriate `srun` command prefixes will cause concurrency within the build.

For more information about GNU parallel `make`, see the `make` manpage. For additional sources of GNU information, see the references provided in "About This Document".

In this section, three different ways to parallelize a make procedure are illustrated. The `smg98` package is used to illustrate these three procedures. The `smg98` package is available at the following URL:

http://www.llnl.gov/asci/applications/SMG98README.html

These procedures take advantage of the GNU `make -j` switch, which specifies the number of jobs to run simultaneously. See the `make` manpage for more information about this switch.

The following parallel make approaches are described:

- "Example Procedure 1" — Go through the directories serially and have the make procedure within each directory be parallel. (Modified Makefile: `Makefile_type1`).
- "Example Procedure 2" — Go through the directories in parallel and have the make procedure within each directory be serial (Modified Makefile: `Makefile_type2`).
- "Example Procedure 3" — Go through the directories in parallel and have the make procedure within each directory be parallel (Modified Makefile: `Makefile_type3`).

The original Makefile is shown below:

```
#BHEADER***********************************************************
# (c) 1998   The Regents of the University of California
#
# See the file COPYRIGHT_and_DISCLAIMER for a complete copyright
# notice, contact person, and disclaimer.
#
# $Revision: 1.1 $
#EHEADER***********************************************************

SHELL = /bin/sh

srcdir = .

HYPRE_DIRS =\
 utilities\
 struct_matrix_vector\
 struct_linear_solvers\
 test

all:
        @ \
        for i in ${HYPRE_DIRS}; \
        do \
          if [ -d $$i ]; \
          then \
            echo "Making $$i ..."; \
            (cd $$i; make); \
            echo ""; \
          fi; \
        done

clean:
        @ \
        for i in ${HYPRE_DIRS}; \
        do \
          if [ -d $$i ]; \
          then \
            echo "Cleaning $$i ..."; \
            (cd $$i; make clean); \
          fi; \
        done

veryclean:
        @ \
        for i in ${HYPRE_DIRS}; \
        do \
          if [ -d $$i ]; \
          then \
            echo "Very-cleaning $$i ..."; \
            (cd $$i; make veryclean); \
          fi; \
        done
```

### 11.3.1 Example Procedure 1

Go through the directories serially and have the make procedure within each directory be parallel.

For the purpose of this exercise we are only parallelizing the “make all” component. The “clean” and “veryclean” components can be parallelized in a similar fashion.

Modified Makefile:

```
all:
        @ \
        for i ${HYPRE_DIRS}; \
        do \
          if [ -d $$i ]; \
          then \
            echo "Making $$i ..."; \
            echo $(PREFIX) $(MAKE) $(MAKE_J) -C $$i; \
            $(PREFIX) $(MAKE) $(MAKE_J) -C $$i;  \
          fi; \
        done
```

By modifying the Makefile to reflect the changes illustrated above, we will now be processing each directory serially and parallelize the individual makes within each directory. The modified `Makefile` is invoked as follows:

```
$ make PREFIX='srun -n1 -N1 MAKE_J='-j4'
```

### 11.3.2 Example Procedure 2

Go through the directories in parallel and have the make procedure within each directory be serial.

For the purpose of this exercise we are only parallelizing the “make all” component. The “clean” and “veryclean” components can be parallelized in a similar fashion.

Modified Makefile:

```
all:

        $(MAKE) $(MAKE_J) struct_matrix_vector/libHYPRE_mv.a

struct_linear_solvers/libHYPRE_ls.a utilities/libHYPRE_utilities.a

        $(PREFIX) $(MAKE) -C test

struct_matrix_vector/libHYPRE_mv.a:

        $(PREFIX) $(MAKE) -C struct_matrix_vector

struct_linear_solvers/libHYPRE_ls.a:

        $(PREFIX) $(MAKE) -C struct_linear_solvers

utilities/libHYPRE_utilities.a:

        $(PREFIX) $(MAKE) -C utilities
```

The modified `Makefile` is invoked as follows:

```
$ make PREFIX='srun -n1 -N1' MAKE_J='-j4'
```

### 11.3.3 Example Procedure 3

Go through the directories in parallel and have the make procedure within each directory be parallel. For the purpose of this exercise, we are only parallelizing the "make all" component. The "clean" and "veryclean" components can be parallelized in a similar fashion.

Modified Makefile:

```
all:

        $(MAKE) $(MAKE_J) struct_matrix_vector/libHYPRE_mv.a

struct_linear_solvers/libHYPRE_ls.a utilities/libHYPRE_utilities.a

        $(PREFIX) $(MAKE) $(MAKE_J) -C test

struct_matrix_vector/libHYPRE_mv.a:

        $(PREFIX) $(MAKE) $(MAKE_J) -C struct_matrix_vector

struct_linear_solvers/libHYPRE_ls.a:

        $(PREFIX) $(MAKE) $(MAKE_J) -C struct_linear_solvers

utilities/libHYPRE_utilities.a:

        $(PREFIX) $(MAKE) $(MAKE_J) -C utilities
```

The modified Makefile is invoked as follows:

```
$ make PREFIX='srun -n1 -N1' MAKE_J='-j4'
```

## 11.4 Local Disks on Compute Nodes

The use of a local disk for private, temporary storage may be configured on the **compute node**s of your HP XC system. Contact your system administrator to find out about the local disks configured on your system.

A local disk is a temporary storage space and does not hold data across execution of applications. Therefore, any information generated by the application during its execution is not saved on the local disk once the application has completed.

## 11.5 I/O Performance Considerations

Before building and running your parallel application, I/O performance issues on the HP XC **cluster** must be considered.

The I/O control system provides two basic types of standard file system views to the application:

- Shared
- Private

### 11.5.1 Shared File View

Although a file opened by multiple processes of an application is shared, each core maintains a private file pointer and file position. This means that if a certain order of input or output from multiple cores is desired, the application must synchronize its I/O requests or position its file pointer such that it acts on the desired file location.

Output requests to standard output and standard error are line-buffered, which can be sufficient output ordering in many cases. A similar effect for other files can be achieved by using append mode when opening the file with the `fopen` system call:

```
fp = fopen ("myfile", "a+");
```

### 11.5.2 Private File View

Although the shared file approach improves ease of use for most applications, some applications, especially those written for shared-nothing clusters, can require the use of file systems private to each node. To accommodate these applications, the system must be configured with local disk.

For example, assume `/tmp` and `/tmp1` have been configured on each compute node.

Now each process can open up a file named `/tmp/myscratch` or `/tmp1/myotherscratch` and each would see a unique file pointer. If these file systems do not exist local to the node, an error results.

It is a good idea to use this option for temporary storage only, and make sure that the application deletes the file at the end.

C example: `fd = open ("/tmp/myscratch", flags)`

Fortran example: `open (unit=9, file="/tmp1/myotherscratch" )`

## 11.6 Communication Between Nodes

On the HP XC system, processes in an **MPI** application run on **compute node**s and use the system interconnect for communication between the nodes. By default, intranode communication is done using shared memory between MPI processes. For information about selecting and overriding the default system interconnect, see the HP-MPI documentation.

## 11.7 Using MPICH on the HP XC System

MPICH is a freely available portable implementation of MPI. For additional information on MPICH, see the following URL:

http://www-unix.mcs.anl.gov/mpi/mpich1/ .

Verify with your system administrator that MPICH has been installed on your system. The *HP XC System Software Administration Guide* provides procedures for setting up MPICH.

MPICH jobs must not run on nodes allocated to other tasks. HP strongly recommends that all MPICH jobs request node allocation through either SLURM or LSF and that MPICH jobs restrict themselves to using only those resources in the allocation.

Launch MPICH jobs using a wrapper script, such as the one shown in Figure 11-1. The following subsections describe how to launch MPICH jobs from a wrapper script with SLURM or LSF, respectively. These subsections are not full solutions for integrating MPICH with the HP XC System Software.

**Figure 11-1 MPICH Wrapper Script**

```
#!/bin/csh
srun csh -c 'echo `hostname`:2' | sort | uniq > machinelist
set hostname = `head -1 machinelist | awk -F: '{print $1}'`
ssh $hostname /opt/mpich/bin/mpirun options... -machinefile machinelist a.out
```

The wrapper script is based on the following assumptions:

- Each node in the HP XC system contains two CPUs.
- The current working directory is available on all nodes on which an MPICH job might run.
- You provide the `mpirun` options that are appropriate to your requirements.
- The executable file is named `a.out`.
- The wrapper script has the appropriate permissions.

You need to modify the wrapper script accordingly if these assumptions are not true.

## 11.7.1 Using MPICH with SLURM Allocation

The SLURM-based allocation method uses the `srun` command to spawn a shell; the remote job is run from within the shell, as shown here:

```
% srun -A options   1
% ./wrapper        2
% exit             3
```

**NOTE:** This method assumes that the communication among nodes is performed using `ssh` and that passwords are not required.

1 The `srun -A` command allocates the resources and spawns a new shell without starting a remote job. For more information on the `-A` option, see *srun*(1) .

**IMPORTANT:** Be sure that the number of nodes and processors in the `srun` command correspond to the numbers specified in the wrapper script.

2 This command line executes the wrapper script to start the job on the allocated nodes.

3 After the MPICH job specified by the wrapper completes, the `exit` command terminates the shell and releases the allocated nodes.

## 11.7.2 Using MPICH with LSF Allocation

The LSF-based allocation method uses a single `bsub` command to create an allocation, as shown here:

```
% bsub -I options... wrapper
```

The `bsub` command launches the wrapper script.

**IMPORTANT:** Be sure that the number of nodes and processors in the `bsub` command corresponds to the number specified by the appropriate options in the wrapper script.

**NOTE:** This method assumes that the communication among nodes is performed using `ssh` and that passwords are not required.

# A Examples

This appendix provides examples that illustrate how to build and run applications on the HP XC system. The examples in this section show you how to take advantage of some of the many methods available, and demonstrate a variety of other user commands to monitor, control, or kill jobs.

The examples in this section assume that you have read the information in previous chapters describing how to use the HP XC commands to build and run parallel applications. Only examples of building and running applications are provided in this section — detailed information about the commands is not provided.

## A.1 Building and Running a Serial Application

This example show how to build a simple application, called `hello world`, and launch it with the SLURM `srun` command.

The following is the source code for the `hello world` program, located in file `hw_hostname.c`.

```
#include <unistd.h>
#include <stdio.h>

int main()
{
  char name[100];
  gethostname(name, sizeof(name));
  printf("%s says Hello!\n", name);
  return 0;
}
```

The `hello world` program is compiled in the usual way:

```
$ cc hw_hostname.c -o hw_hostname
```

When run on the login node, it shows the name of the login node; `n16` in this case:

```
$ ./hw_hostname
n16 says Hello!
```

The `srun` command can be used to run it on one of the **compute node**s in the default SLURM partition. This time it lands on `n13`:

```
$ srun ./hw_hostname
n13 says Hello!
```

The `srun` command can also be used to replicate it on several compute nodes. This is not generally useful, but is included for illustrative purposes.

```
$ srun -n4 ./hw_hostname
n13 says Hello!
n13 says Hello!
n14 says Hello!
n14 says Hello!
```

## A.2 Launching a Serial Interactive Shell Through LSF

This section provides an example that shows how to launch a serial interactive shell through LSF. The `bsub -Is` command is used to launch an interactive shell through LSF. This example steps through a series of commands that illustrate what occurs when you launch an interactive shell.

### Examine the LSF execution host information:

```
$ bhosts
HOST_NAME            STATUS JL/U MAX NJOBS RUN SSUSP USUSP RSV
lsfhost.localdomain  ok       -   12     0   0     0     0   0
```

Examine the partition information:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     6   idle n[5-10]
```

Examine the local host information:

```
$ hostname
n2
```

Examine the job information:

```
$ bjobs
No unfinished job found
```

Run the LSF `bsub -Is` command to launch the interactive shell:

```
$ bsub -Is -n1 /bin/bash
Job <120> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
```

Note the output when `hostname` is again issued:

```
$ hostname
n16
```

Note the FROM_HOST when `bjobs` is again issued:

```
$ bjobs
JOBID USER  STAT QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME SUBMIT_TIME
120   smith RUN  normal n2        lsfhost.loc /bin/bash date and time
```

Note the output when `srun hostname` is issued:

```
$ srun hostname
n10
```

Exit from the shell:

```
$ exit
exit
```

## A.3 Running LSF Jobs with a SLURM Allocation Request

### A.3.1 Example 1. Two Cores on Any Two Nodes

This example submits a job that requests two cores on any two nodes, on an HP XC system that has three compute nodes.

Submit the job:

```
$ bsub -n2 -ext "SLURM[nodes=2]" -I srun hostname
Job <8> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n13
n14
```

View the job:

```
$ bjobs -l 8
Job <8>, User <smith>, Project <default>, Status <DONE>, Queue <normal>,
Interactive mode, Extsched <SLURM[nodes=2]>, Command <srun hostname>
```

```
date and time stamp: Submitted from host <lsfhost.localdomain>,
                     CWD <$HOME>, 2 Processors Requested;
date and time stamp: Started on 2 Hosts/Processors
                     <2*lsfhost.localdomain>;
date and time stamp: slurm_id=24;ncpus=4;slurm_alloc=n[13-14];
date and time stamp: 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       lsfadmin  date and time   SLURM[nodes=2]   N
```

## A.3.2 Example 2. Four Cores on Two Specific Nodes

This example submits a job that requests four cores on two specific nodes, on an XC system that has three compute nodes.

Submit the job:

```
$ bsub -I -n4 -ext "SLURM[nodelist=n[14,16]]" srun hostname
Job <9> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
n14
n14
n16
n16
```

View the job:

```
$ bjobs -l 9
Job <9>, User <smith>, Project <default>, Status <DONE>,
Queue <normal>, Interactive mode, Extsched <SLURM[nodelist=n[14,16]]>,
Command <srun hostname>

date and time stamp: Submitted from host <lsfhost.localdomain>,
                     CWD <$HOME>, 4 Processors Requested;
date and time stamp: Started on 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=24;ncpus=4;<SLURM[nodelist=n[14,16]]>;
date and time stamp: 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       lsfadmin  date and time  SLURM[nodelist=n[14,16]]    N
```

## A.4 Launching a Parallel Interactive Shell Through LSF

This section provides an example that shows how to launch a parallel interactive shell through LSF. The `bsub -Is` command is used to launch an interactive shell through LSF. This example steps through a series of commands that illustrate what occurs when you launch an interactive shell.

### Examine the LSF execution host information:

```
$ bhosts
HOST_NAME           STATUS  JL/U  MAX  NJOBS RUN SSUSP USUSP RSV
lsfhost.localdomain ok       -    12     0    0   0    0     0
```

### Examine the partition information:

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     6   idle n[5-10]
```

### Examine the local host information:

```
$ hostname
n2
```

### Examine the job information:

```
$ bjobs
No unfinished job found
```

### Run the LSF `bsub -Is` command to launch the interactive shell:

```
$ bsub -Is -n4 -ext "SLURM[nodes=4]" /bin/bash
Job <124> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
```

### Note the output when `hostname` is again issued:

```
$ hostname
n16
```

### Note the output when `srun hostname` is issued:

```
$ srun hostname
n5
n7
n6
n8
```

### Note the output from the `bjobs` command:

```
$ bjobs
JOBID USER  STAT QUEUE  FROM_HOST EXEC_HOST     JOB_NAME  SUBMIT_TIME
124   lsfad RUN  normal n2        4*lsfhost.loc /bin/bash date and time
```

### Examine the running job's information:

```
$ bhist -l 124
Job <124>, User <lsfadmin>, Project <default>,
Interactive pseudo-terminal shell mode,
Extsched <SLURM[nodes=4]>, Command </bin/bash>
date and time stamp: Submitted from host <n2>,
to Queue <normal>, CWD <$HOME>,4 Processors Requested,
Requested Resources <type=any>;

date and time stamp: Dispatched to 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];
date and time stamp: Starting (Pid 4785);
```

```
Summary of time in seconds spent in various states by date and time
  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
  11     0     124    0      0      0     135
```

### Exit from the shell:

```
$ exit
exit
```

### Examine the finished job's information:

```
$ bhist -l 124
Job <124>, User <lsfadmin>, Project <default>,
Interactive pseudo-terminal shell mode,
Extsched <SLURM[nodes=4]>, Command </bin/bash>
date and time stamp: Submitted from host <n2>,
to Queue <normal>, CWD <$HOME>,
4 Processors Requested, Requested Resources <type=any>;

date and time stamp: Dispatched to 4 Hosts/Processors
                     <4*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=8;slurm_alloc=n[5-8];
date and time stamp: Starting (Pid 4785);
date and time stamp: Done successfully.
The CPU time used is 0.1 seconds;
date and time stamp: Post job process done successfully;

Summary of time in seconds spent in various states by date and time
  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
  11      0    220   0       0      0     231
```

## A.5 Submitting a Simple Job Script with LSF

This example submits a job script (`myjobscript.sh`) with the `bsub -I` option. Inside the script, there are two `srun` commands. The first command displays the host name and second command displays system information.

In this example, the run-time environment is first explored. Next, the contents of the `myjobscript.sh` script is displayed. Then an example of a command to launch the script is shown. Finally, the resulting output of the script is provided.

### Show the environment:

```
$ lsid
Platform LSF HPC version, Update n, build date stamp
Copyright 1992-2008 Platform Computing Corporation

My cluster name is penguin
My master name is lsfhost.localdomain
$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
lsf        up     infinite   4      alloc  n[13-16]
$ lshosts
HOST_NAME    type     model    cpuf ncpus maxmem maxswp server RESOURCES
lsfhost.loc  SLINUX6  DEFAULT 1.0    8    1M      -    Yes    (slurm)
$ bhosts
HOST_NAME    STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
lsfhost.loc  ok       -     8     0     0     0      0     0
```

### Display the script:

```
$ cat myjobscript.sh
#!/bin/sh
```

```
srun hostname
srun uname -a
```

### Run the job:

```
$ bsub -I -n4 myjobscript.sh
Job <1006> is submitted to default queue <normal>.
<<Waiting for dispatch>>
<<Starting on lsfhost.localdomain>>
n14
n14
n16
n16
Linux n14 2.4.21-15.3hp.XCsmp #2 SMP date and time stamp
                                     ia64 ia64 ia64 GNU/Linux
Linux n14 2.4.21-15.3hp.XCsmp #2 SMP date and time stamp
                                     ia64 ia64 ia64 GNU/Linux
Linux n16 2.4.21-15.3hp.XCsmp #2 SMP date and time stamp
                                     ia64 ia64 ia64 GNU/Linux
Linux n16 2.4.21-15.3hp.XCsmp #2 SMP date and time stamp
                                     ia64 ia64 ia64 GNU/Linux
```

## A.6 Submitting an Interactive Job with LSF

This example shows how to submit a batch interactive job to LSF with the `bsub -Ip` command. When you specify the `-Ip` option, `bsub` submits a batch interactive job and creates a pseudo-terminal when the job starts. A new job cannot be submitted until the interactive job is completed or terminated.

### Show the environment:

```
$ lsid
Platform LSF HPC version, Update n, build date stamp
Copyright 1992-2008 Platform Computing Corporation

My cluster name is penguin
My master name is lsfhost.localdomain
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4   idle n[13-16]
$ lshosts
HOST_NAME   type     model    cpuf ncpus maxmem maxswp server RESOURCES
lsfhost.loc SLINUX6  DEFAULT  1.0    8    1M      -    Yes    (slurm)
$ bhosts
HOST_NAME    STATUS   JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
lsfhost.loc  ok        -     8     0     0     0      0     0
$ sinfo
PARTITION AVAIL TIMELIMIT NODES  STATE NODELIST
lsf          up  infinite     4   idle n[13-16]
```

### Submit the job:

```
$ bsub -n8 -Ip /bin/sh
Job <1008> is submitted to default queue <normal>.
<<Waiting for dispatch>>
<<Starting on lsfhost.localdomain>>
```

### Show the job allocation:

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

## Show the SLURM job ID:

```
$ env | grep SLURM
SLURM_JOBID=74
SLURM_NPROCS=8
```

## Run some commands from the pseudo-terminal:

```
$ srun hostname
n13
n13
n14
n14
n15
n15
n16
n16
```

```
$ srun -n3 hostname
n13
n14
n15
```

## Exit the pseudo-terminal:

```
$ exit
exit
```

## View the interactive jobs:

```
$ bjobs -l 1008
Job <1008>, User smith, Project <default>, Status <DONE>, Queue <normal>,
Interactive pseudo-terminal mode, Command </bin/sh>

date and time stamp: Submitted from host n16,
                     CWD <$HOME/tar_drop1/test>, 8 Processors Requested;
date and time stamp: Started on 8Hosts/Processors<8*lsfhost.localdomain>;
date and time stamp: slurm_id=74;ncpus=8;slurm_alloc=n16,n14,n13,n15;
date and time stamp: Done successfully. The CPU  time used is 0.1 seconds.


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

## View the finished jobs:

```
$ bhist -l 1008
Job <1008>, User smith, Project <default>,
Interactive pseudo-terminal mode, Command </bin/sh>

date and time stamp: Submitted from host n16, to Queue <normal>,
                     CWD <$HOME/tar_drop1/test>, 8 Processors Requested;
date and time stamp: Dispatched to 8 Hosts/Processors
                     <8*lsfhost.localdomain>;
date and time stamp: slurm_id=74;ncpus=8;slurm_alloc=n16,n14,n13,n15;
date and time stamp: Starting (Pid 26446);
date and time stamp: Done successfully. The CPU time used is 0.1 seconds;
date and time stamp: Post job process done successfully;

Summary of time in seconds spent in various states by date and time
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
  12       0        93       0        0        0        105
```

### View the node state:

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

## A.7 Submitting an HP-MPI Job with LSF

This example shows how to run an **MPI** job with the `bsub` command.

### Show the environment:

```
$ lsid
Platform LSF HPC version, Update n, build date stamp
Copyright 1992-2008 Platform Computing Corporation

My cluster name is penguin
My master name is lsfhost.localdomain
$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
lsf        up     infinite   4      alloc  n[13-16]
$ lshosts
HOST_NAME    type     model    cpuf ncpus maxmem maxswp server RESOURCES
lsfhost.loc  SLINUX6  DEFAULT 1.0    8    1M      -    Yes    (slurm)
$ bhosts
HOST_NAME          STATUS  JL/U  MAX NJOBS RUN SSUSP USUSP RSV
lsfhost.localdomai  ok      -     8     0   0    0     0    0
```

### Run the job:

```
$ bsub -I -n6 -ext "SLURM[nodes=3]" mpirun -srun /usr/share/hello
Job <1009> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
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
```

### View the running job:

```
$ bjobs -l 1009
Job <1009>, User <smith>, Project <default>,
Status <DONE>, Queue <normal>,
Interactive mode, Extsched <SLURM[nodes=3]>,
Command </opt/hpmpi/bin/mpirun -srun /usr/share/hello>

date and time stamp: Submitted from host <lsfhost.localdomain>,
                     CWD <$HOME>, 6 Processors Requested;
date and time stamp: Started on 6 Hosts/Processors
                     <6*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=6;slurm_alloc=n[13-15];
date and time stamp: 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       lsfadmin  date and time   SLURM[nodes=2]   N
```

### View the finished job:

```
$ bhist -l 1009
Job <1009>, User <smith>, Project <default>,
Interactive mode, Extsched <SLURM[nodes=3]>,
Command </opt/hpmpi/bin/mpirun -srun /usr/share/hello>

date and time stamp: Submitted from host <lsfhost.localdomain>,
                     to Queue <normal>,CWD <$HOME>,
                     6 Processors Requested;
date and time stamp: Dispatched to 6 Hosts/Processors
                     <6*lsfhost.localdomain>;
date and time stamp: slurm_id=22;ncpus=6;slurm_alloc=n[13-15];
date and time stamp: Starting (Pid 11216);
date and time stamp: Done successfully.
The CPU time used is 0.0 seconds;
date and time stamp: Post job process done successfully;

Summary of time in seconds spent in various states by date and time
  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
  11    0      7    0      0      0      18
```

## A.8 Using a Resource Requirements String in an LSF Command

The following examples show how to use a resource requirement string in an LSF command.

- Run `myjob` on an Alpha/AXP host or an HP XC host if one is available, but not both:

  ```
  $ bsub -n 8 -R "ALPHA5 || SLINUX64" \
  -ext "SLURM[nodes=4-4]" myjob
  ```

  If `myjob` runs on an HP XC host, the `SLURM[nodes=4-4]` allocation option is applied. If it runs on an Alpha/AXP 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 HP XC host, the `SLURM[nodes=4-4]` option is applied. If `myjob` runs on an HP AlphaServer SC host, the RMS `ptile` option is applied. If it runs on any other host type, the SLURM and RMS options are ignored.

# Glossary

## A

| | |
|---|---|
| **administration branch** | The half (branch) of the administration network that contains all of the general-purpose administration ports to the nodes of the HP XC system. |
| **administration network** | The private network within the HP XC system that is used for administrative operations. |
| **availability set** | An association of two individual nodes so that one node acts as the first server and the other node acts as the second server of a service.<br>*See also* improved availability, availability tool. |
| **availability tool** | A software product that enables system services to continue running if a hardware or software failure occurs by failing over the service to the other node in an availability set.<br>*See also* improved availability, availability set. |

## B

| | |
|---|---|
| **base image** | The collection of files and directories that represents the common files and configuration data that are applied to all nodes in an HP XC system. |
| **branch switch** | A component of the Administration Network. A switch that is uplinked to the root switch and receives physical connections from multiple nodes. |

## C

| | |
|---|---|
| **cluster** | A set of independent computers combined into a unified system through system software and networking technologies. |
| **cluster alias** | The external cluster host name supported by LVS, which enables inbound connections without having to know individual nodes names to connect and log in to the HP XC system. |
| **CMDB** | Configuration and management database. Constructed during HP XC system installation, the CMDB is a MySQL database that stores information about the nodes, hardware, and software configuration, and network connectivity. This database runs on the node with the node management role. |
| **compute node** | A node that is assigned only with the compute role and no other. Jobs are distributed to and run on nodes with the `compute` role; no other services run on a compute node.<br>. |
| **configuration and management database** | *See* CMDB. |
| **console branch** | A component of the administration network. The half (branch) of the administration network that contains all of the console ports of the nodes of the HP XC system. This branch is established as a separate branch to enable some level of partitioning of the administration network to support specific security needs. |

## D

| | |
|---|---|
| **DHCP** | Dynamic Host Control Protocol. A protocol that dynamically allocates IP addresses to computers on a local area network. |
| **Dynamic Host Control Protocol** | *See* DHCP. |

## E

| | |
|---|---|
| **EFI** | Extensible Firmware Interface. Defines a model for the interface between operating systems and Itanium-based platform firmware. The interface consists of data tables that contain platform-related information, plus boot and run-time service calls that are available to the |

| | |
|---|---|
| | operating system and its loader. Together, these provide a standard environment for booting an operating system and running preboot applications. |
| **enclosure** | The hardware and software infrastructure that houses HP BladeSystem servers. |
| **extensible firmware interface** | *See* EFI. |
| **external network node** | A node that is connected to a network external to the HP XC system. |

F

| | |
|---|---|
| **fairshare** | An LSF job-scheduling policy that specifies how resources should be shared by competing users. A fairshare policy defines the order in which LSF attempts to place jobs that are in a queue or a host partition. |
| **FCFS** | First-come, first-served. An LSF job-scheduling policy that specifies that jobs are dispatched according to their order in a queue, which is determined by job priority, not by order of submission to the queue. |
| **first-come, first-served** | *See* FCFS. |

G

| | |
|---|---|
| **global storage** | Storage within the HP XC system that is available to all of the nodes in the system. Also known as local storage. |
| **golden client** | The node from which a standard file system image is created. The golden image is distributed by the image server. In a standard HP XC installation, the head node acts as the image server and golden client. |
| **golden image** | A collection of files, created from the golden client file system that are distributed to one or more client systems. Specific files on the golden client may be excluded from the golden image if they are not appropriate for replication. |
| **golden master** | The collection of directories and files that represents all of the software and configuration data of an HP XC system. The software for any and all nodes of an HP XC system can be produced solely by the use of this collection of directories and files. |

H

| | |
|---|---|
| **head node** | The single node that is the basis for software installation, system configuration, and administrative functions in an HP XC system. There may be another node that can provide a failover function for the head node, but HP XC system has only one head node at any one time. |
| **host name** | The name given to a computer. Lowercase and uppercase letters (a–z and A–Z), numbers (0–9), periods, and dashes are permitted in host names. Valid host names contain from 2 to 63 characters, with the first character being a letter. |

I

| | |
|---|---|
| **I/O node** | A node that has more storage available than the majority of server nodes in an HP XC system. This storage is frequently externally connected storage, for example, SAN attached storage. When configured properly, an I/O server node makes the additional storage available as global storage within the HP XC system. |
| **iLO** | Integrated Lights Out. A self-contained hardware technology available on CP3000 and CP4000 cluster platform hardware models that enables remote management of any node within a system. |
| **iLO2** | The next generation of iLO that provides full remote graphics console access and remote virtual media.<br>*See also* iLO. |

| | |
|---|---|
| **image server** | A node specifically designated to hold images that will be distributed to one or more client systems. In a standard HP XC installation, the head node acts as the image server and golden client. |
| **improved availability** | A service availability infrastructure that is built into the HP XC system software to enable an availability tool to fail over a subset of eligible services to nodes that have been designated as a second server of the service<br>*See also* availability set, availability tool. |
| **Integrated Lights Out** | *See* iLO. |
| **interconnect** | A hardware component that provides high-speed connectivity between the nodes in the HP XC system. It is used for message passing and remote memory access capabilities for parallel applications. |
| **interconnect module** | A module in an HP BladeSystem server. The interconnect module provides the Physical I/O ports for the server blades and can be either a switch, with connections to each of the server blades and some number of external ports, it can be or a pass-through module, with individual external ports for each of the server blades.<br>*See also* server blade. |
| **interconnect network** | The private network within the HP XC system that is used primarily for user file access and for communications within applications. |
| **Internet address** | A unique 32-bit number that identifies a host's connection to an Internet network. An Internet address is commonly represented as a network number and a host number and takes a form similar to the following: `192.0.2.0`. |
| **IPMI** | Intelligent Platform Management Interface. A self-contained hardware technology available on HP ProLiant DL145 servers that enables remote management of any node within a system. |
| **ITRC** | HP IT Resource Center. The HP corporate Web page where software patches are made available. The web address is http://www.itrc.hp.com. To download patches from this Web page, you must register as an Americas/Asia Pacific or European customer. |

## L

| | |
|---|---|
| **Linux Virtual Server** | *See* LVS. |
| **load file** | A file containing the names of multiple executables that are to be launched simultaneously by a single command. |
| **Load Sharing Facility** | *See* LSF with SLURM. |
| **local storage** | Storage that is available or accessible from one node in the HP XC system. |
| **LSF execution host** | The node on which LSF runs. A user's job is submitted to the LSF execution host. Jobs are launched from the LSF execution host and are executed on one or more compute nodes. |
| **LSF master host** | The overall LSF coordinator for the system. The master load information manager (LIM) and master batch daemon (`mbatchd`) run on the LSF master host. Each system has one master host to do all job scheduling and dispatch. If the master host goes down, another LSF server in the system becomes the master host. |
| **LSF with SLURM** | Load Sharing Facility integrated with SLURM. The batch system resource manager on an HP XC system that is integrated with SLURM. LSF with SLURM places a job in a queue and allows it to run when the necessary resources become available. LSF with SLURM manages just one resource: the total number of processors designated for batch processing.<br><br>LSF with SLURM can also run interactive batch jobs and interactive jobs. An LSF interactive batch job allows you to interact with the application while still taking advantage of LSF with SLURM scheduling policies and features. An LSF with SLURM interactive job is run without using LSF with SLURM batch processing features but is dispatched immediately by LSF with SLURM on the LSF execution host.<br><br>*See also* LSF execution host. |

| | |
|---|---|
| **LVS** | Linux Virtual Server. Provides a centralized login capability for system users. LVS handles incoming login requests and directs them to a node with a login role. |

## M

| | |
|---|---|
| **Management Processor** | *See* MP. |
| **master host** | *See* LSF master host. |
| **MCS** | An optional integrated system that uses chilled water technology to triple the standard cooling capacity of a single rack. This system helps take the heat out of high-density deployments of servers and blades, enabling greater densities in data centers. |
| **Modular Cooling System** | *See* MCS. |
| **module** | A package that provides for the dynamic modification of a user's environment by means of modulefiles.<br>*See also* modulefile. |
| **modulefile** | Contains information that alters or sets shell environment variables, such as `PATH` and `MANPATH`. Modulefiles enable various functions to start and operate properly. |
| **MP** | Management Processor. Controls the system console, reset, and power management functions on HP Integrity servers. |
| **MPI** | Message Passing Interface. A library specification for message passing, proposed as a standard by a broadly based committee of vendors, implementors, and users. |
| **MySQL** | A relational database system developed by MySQL AB that is used in HP XC systems to store and track system configuration information. |

## N

| | |
|---|---|
| **NAT** | Network Address Translation. A mechanism that provides a mapping (or transformation) of addresses from one network to another. This enables external access of a machine on one LAN that has the same IP address as a machine on another LAN, by mapping the LAN address of the two machines to different external IP addresses. |
| **Network Address Translation** | *See* NAT. |
| **Network Information Services** | *See* NIS. |
| **NIS** | Network Information Services. A mechanism that enables centralization of common data that is pertinent across multiple machines in a network. The data is collected in a domain, within which it is accessible and relevant. The most common use of NIS is to maintain user account information across a set of networked hosts. |
| **NIS client** | Any system that queries NIS servers for NIS database information. Clients do not store and maintain copies of the NIS maps locally for their domain. |
| **NIS master server** | A system that stores the master copy of the NIS database files, or maps, for the domain in the `/var/yp/DOMAIN` directory and propagates them at regular intervals to the slave servers. Only the master maps can be modified. Each domain can have only one master server. |
| **NIS slave server** | A system that obtains and stores copies of the master server's NIS maps. These maps are updated periodically over the network. If the master server is unavailable, the slave servers continue to make the NIS maps available to client systems. Each domain can have multiple slave servers distributed throughout the network. |

## O

| | |
|---|---|
| **OA** | The enclosure management hardware, software, and firmware that is used to support all of the managed devices contained within the HP BladeSystem c-Class enclosure. |

| | |
|---|---|
| **onboard administrator** | *See* OA. |

## P

| | |
|---|---|
| **parallel application** | An application that uses a distributed programming model and can run on multiple processors. An HP XC MPI application is a parallel application. That is, all interprocessor communication within an HP XC parallel application is performed through calls to the MPI message passing library. |
| **PXE** | Preboot Execution Environment. A standard client/server interface that enables networked computers that are not yet installed with an operating system to be configured and booted remotely. PXE booting is configured at the BIOS level. |

## R

| | |
|---|---|
| **remote graphics software** | *See* RGS. |
| **resource management role** | Nodes with this role manage the allocation of resources to user applications. |
| **RGS** | HP Remote Graphics Software. A utility that enables remote access and sharing of a graphics workstation desktop. |
| **role** | A set of services that are assigned to a node. |
| **Root Administration Switch** | A component of the administration network. The top switch in the administration network; it may be a logical network switch comprised of multiple hardware switches. The Root Console Switch is connected to the Root Administration Switch. |
| **root node** | A node within an HP XC system that is connected directly to the Root Administration Switch. |
| **RPM** | Red Hat Package Manager.<br>**1.** A utility that is used for software package management on a Linux operating system, most notably to install and remove software packages.<br>**2.** A software package that is capable of being installed or removed with the RPM software package management utility. |

## S

| | |
|---|---|
| **scalable visualization array** | *See* SVA. |
| **serial application** | A command or user program that does not use any distributed shared-memory form of parallelism. A serial application is basically a single-processor application that has no communication library calls (for example, MPI, PVM, GM, or Portals).<br>An example of a serial application is a standard Linux command, such as the `ls` command. Another example of a serial application is a program that has been built on a Linux system that is binary compatible with the HP XC environment, but does not contain any of the HP XC infrastructure libraries. |
| **server blade** | One of the modules of an HP BladeSystem. The server blade is the compute module consisting of the CPU, memory, I/O modules and other supporting hardware. Server blades do not contain their own physical I/O ports, power supplies, or cooling. |
| **SLURM backup controller** | The node on which the optional backup `slurmctld` daemon runs. On SLURM failover, this node becomes the SLURM master controller. |
| **SLURM master controller** | The node on which the `slurmctld` daemon runs. |
| **SMP** | Symmetric multiprocessing. A system with two or more CPUs that share equal (symmetric) access to all of the facilities of a computer system, such as the memory and I/O subsystems. In |

| | |
|---|---|
| | an HP XC system, the use of SMP technology increases the number of CPUs (amount of computational power) available per unit of space. |
| **ssh** | Secure Shell. A shell program for logging in to and executing commands on a remote computer. It can provide secure encrypted communications between two untrusted hosts over an insecure network. |
| **standard LSF** | A workload manager for any kind of batch job. Standard LSF features comprehensive workload management policies in addition to simple first-come, first-serve scheduling (fairshare, preemption, backfill, advance reservation, service-level agreement, and so on). Standard LSF is suited for jobs that do not have complex parallel computational needs and is ideal for processing serial, single-process jobs. Standard LSF is not integrated with SLURM. |
| **SVA** | HP Scalable Visualization Array. A highly affordable, scalable, ready-to-run visualization solution that completes the HP Unified Cluster Portfolio's integration of computation, data management and visualization in a single, integrated cluster environment. The HP SVA solution adds high-performance HP workstations in building block configurations that combine with industry-standard visualization components. State-of-the-art industry standard and open source clustering, graphics, and networking technology are leveraged to reduce costs and enhance flexibility. The tight integration of scalable computation, data management and visualization enables the following: clustered parallel visualization applications with support for very large data sets, display of complex, high resolution images, including volume visualization, and real-time rendering with computational steering through closely coupled visualization, computation and data management. |
| **symmetric multiprocessing** | *See* SMP. |

# Index