Managing System Software for the Cray® Linux Environment

S–2393–5201

RECORD OF REVISION

S–2393–5201 Published June 2014 Supports the Cray Linux Environment (CLE) 5.2.UP01 release and the System Management Workstation (SMW) 7.2.UP01 release for Cray XC30 and Cray XC30-AC systems.

S–2393–52xc Published March 2014 Supports the Cray Linux Environment (CLE) 5.2.UP00 release and the System Management Workstation (SMW) 7.2.UP00 release for Cray XC30 and Cray XC30-AC systems.

S–2393–5101 Published December 2013 Supports the Cray Linux Environment (CLE) 5.1.UP01 release and the System Management Workstation (SMW) 7.1.UP01 release.

S–2393–4202 Published October 2013 Supports the Cray Linux Environment (CLE) 4.2.UP02 release and the System Management Workstation (SMW) 7.1.UP00 release.

S–2393–51 Published September 2013 Supports the Cray Linux Environment (CLE) 5.1.UP00 release and the System Management Workstation (SMW) 7.1.UP00 release.

S–2393–4201 Published July 2013 Supports the Cray Linux Environment (CLE) 4.2.UP01 release and the System Management Workstation (SMW) 7.0.UP03 release.

S–2393–5003 Published June 2013 Supports the Cray Linux Environment (CLE) 5.0.UP03 release and the System Management Workstation (SMW) 7.0.UP03 release.

S–2393–42 Published April 2013 Supports the Cray Linux Environment (CLE) 4.2 release and the System Management Workstation (SMW) 7.0.UP02 release.

S–2393–5002 Published March 2013 Supports the Cray Linux Environment (CLE) 5.0.UP02 release and the System Management Workstation (SMW) 7.0.UP02 release.

S–2393–4101 Published December 2012 Supports the Cray Linux Environment (CLE) 4.1.UP01 release and the System Management Workstation (SMW) 7.0.UP01 release.

S–2393–5001 Published November 2012 Limited Availability (LA) draft; supports the Cray Linux Environment (CLE) 5.0.UP01 release and the System Management Workstation (SMW) 7.0.UP01 release.

S–2393–41 Published August 2012 Limited Availability (LA) draft; supports the Cray Linux Environment (CLE) 4.1.UP00 LA release and the System Management Workstation (SMW) 7.0.UP00 LA release.

S–2393–4003 Published March 2012 Supports the Cray Linux Environment (CLE) 4.0.UP03 release and the System Management Workstation (SMW) 6.0.UP03 release.

S–2393–4002 Published December 2011 Supports the Cray Linux Environment (CLE) 4.0.UP02 release and the System Management Workstation (SMW) 6.0.UP02 release.

S–2393–4001 Published September 2011 Supports the Cray Linux Environment (CLE) 4.0.UP01 release and the System Management Workstation (SMW) 6.0.UP01 release.

S–2393–4001 Published August 2011 Limited Availability draft; supports the Cray Linux Environment (CLE) 4.0.UP00 release and the System Management Workstation (SMW) 6.0.UP00 release.

S–2393–3102 Published December 2010 Supports the Cray Linux Environment (CLE) 3.1.02 release and the System Management Workstation (SMW) 5.1.02 release.

3.1 Published June 2010 Supports the Cray Linux Environment (CLE) 3.1 release and the System Management Workstation (SMW) 5.1 release.

3.0 Published March 2010 Supports the Cray Linux Environment (CLE) 3.0 release and the System Management Workstation (SMW) 5.0 release.

2.2 Published July 2009 Supports the general availability (GA) release of Cray XT systems running the Cray Linux Environment (CLE) 2.2 release and the general availability (GA) release of the System Management Workstation (SMW) 4.0 release.

2.1 Published November 2008 Supports the general availability (GA) release of Cray XT systems running the Cray Linux Environment (CLE) 2.1 release and the System Management Workstation (SMW) 3.1 release as of the SMW 3.1.09 update.

2.0 Published October 2007 Supports general availability (GA) release of Cray XT series systems running the Cray XT series Programming Environment 2.0, UNICOS/lc 2.0, and System Management Workstation 3.0.1 releases.

1.5 Published October 2006 Supports general availability (GA) release of Cray XT series systems running the Cray XT series Programming Environment 1.5, UNICOS/lc 1.5, and System Management Workstation 1.5 releases.

1.4 Published May 2006 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.4, Cray XT3 RAS and Management System (CRMS) 1.4, and UNICOS/lc 1.4 releases.

1.3 Published November 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.3, System Management Workstation (SMW) 1.3, and UNICOS/lc 1.3 releases.

1.2 Published September 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.2, System Management Workstation (SMW) 1.2, and UNICOS/lc 1.2 releases.

1.1 Published June 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.1, System Management Workstation (SMW) 1.1, and UNICOS/lc 1.1 releases.

1.0 Published February 2005 Draft documentation to support Cray XT3 limited availability (LA) systems.

Changes to this Document

S–2393–5201

Added information

- Validating the Health of the HSS on page 106.
- Added the CCM plugin test to Configuring Node Health Checker Tests on page 162.
- Adding or Removing a High-speed Network Cable from Service on page 218.
- RUR `kncstats` on page 355 and `taskstats` on page 357.
- Extended data option to RUR `energy` on page 352.
- Output formatting options to RUR `energy` on page 352 and `taskstats` on page 357.
- User opt-in option to RUR `user` on page 361.
- New configuration parameters to enable suspend/resume added to The `alps.conf` Configuration File on page 275.

Revised information

- Procedure 54 on page 219.
- Procedure 55 on page 220.

Deleted information

- Listing of the example configuration file from RUR Configuration File on page 350.

S–2393–52xc

Added information

- Chapter 13, Configuring Service Node MAMU on page 375
- Extended Node ID (XNID) on page 58

Revised information

- Updated step 2 of Procedure 51.

Deleted information

- Cray Audit is deprecated and the "Security Auditing and Cray Audit Extensions" section was removed.

S–2393–5101

Added information

- Added Resource Utilization Reporting (RUR) plugins `timestamp`, `exit_code`, and `user_output` to Plugin Architecture on page 349.

S–2393–51

Revised information

- State Manager on page 48: The state manager uses the Lightweight Log Manager (LLM). The log data from state manager is written to `/var/opt/cray/log/sm-`*yyyymmdd*. The default setting for state manager is to enable LLM logging.
- Updated naming conventions in Table 1.
- Procedure 50 on page 214 includes additional steps and updated Note.
- Increasing the Boot Manager Time-out Value on page 259 shows revised `/opt/cray/hss/default/etc/bm.ini` `boot_timeout` values, depending on system size.

Deleted information

- `ALPS_NIDORDER` was removed from `/etc/sysconfig/alps` Configuration File on page 269 as the tuning associated with this parameter is no longer relevant to the Cray XC30 system topology.

Contents

Page

Page

Page

Page

Page

Page

Page

Page

Page

Page

Procedures

Page

Page

Page

Examples

Page

Page

Page

Page

Figures

Tables

Page

Introduction [1]

The release documents provided with your Cray Linux Environment (CLE) operating system and Cray System Management Workstation (SMW) release packages state the specific Cray platforms supported with each release package.

A Cray system is a massively parallel processing (MPP) system that has a shared-root file system available to all service-processing elements nodes). Cray has combined commodity and open-source components with custom-designed components to create a system that can operate efficiently at immense scale.

The Cray Linux Environment (CLE) operating system includes Cray's customized version of the SUSE Linux Enterprise Server (SLES) 11 Service Pack 3 (SP3) operating system, with a Linux 3.0.93 kernel. This full-featured operating system runs on the Cray system's service nodes. Service nodes perform the functions needed to support users, administrators, and applications running on compute nodes. Above the operating system level are specialized daemons and applications that perform functions unique to each service node.

Compute nodes on Cray systems run the *CNL* compute node operating system, which runs a Linux kernel. The kernel provides support for application execution without the overhead of a full operating-system image. The kernel interacts with an application process in very limited ways. It provides virtual memory addressing and physical memory allocation, memory protection, access to the message-passing layer, and a scalable job loader. Support for I/O operations is limited inside the compute node's kernel. For a more complete description, see Compute Nodes on page 42.

Note: Functionality marked as deferred in this documentation is planned to be implemented in a later release.

1.1 Audience for This Guide

The audience for this guide is system administrators and those who manage the operation of a Cray system. Prerequisites for using this guide include a working knowledge of Linux to administer the system and a review of the Cray system administration documentation listed in Cray System Administration Publications and in Related Publications on page 30, of this guide. This guide assumes that you have a basic understanding of your Cray system and the software that runs on it.

1.2 Cray System Administration Publications

This publication is one of a set of related manuals that cover information about the structure and operation of your Cray system. See also:

- *Installing Cray System Management Workstation (SMW) Software* (S–2480)
- *Cray System Management Workstation (SMW) Software Release Errata*
- *Cray System Management Workstation (SMW) Software Release README file*
- *Cray System Management Workstation (SMW) Software Release Notes file*
- *Cray Linux Environment (CLE) Software Release Overview* (S–2425)
- *Cray Linux Environment (CLE) Software Release Overview Supplement* (S–2497)
- *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444)
- *Limitations for the CLE Release*
- *CLE Release Errata*
- *Managing Lustre for the Cray Linux Environment (CLE)* (S–0010)
- *Introduction to Cray Data Virtualization Service* (S–0005)
- *Configuring SEC Software for a Cray XC, Cray XE, or Cray XK System* (S–2542)
- *Network Resiliency for Cray XC30 Systems* (S–0041)
- *Using and Configuring System Environment Data Collections (SEDC)* (S–2491)
- *Using the GNI and DMAPP APIs* (S–2446)
- *Workload Management and Application Placement for the Cray Linux Environment* (S–2496)
- *Writing a Node Health Checker (NHC) Plugin Test* (S–0023)
- *Using Cray Performance Measurement and Analysis Tools* (S–2376)
- *Cray Programming Environment User's Guide* (S–2529)
- *Cray Programming Environments Installation Guide* (S–2372)
- *Modifying Your Application to Avoid Aries Network Congestion* (S–0048)

1.3 Related Publications

Because your Cray system runs a combination of software developed by Cray, other vendors' software, and open-source software, the following websites may be useful:

- Linux Documentation Project — See http://www.tldp.org

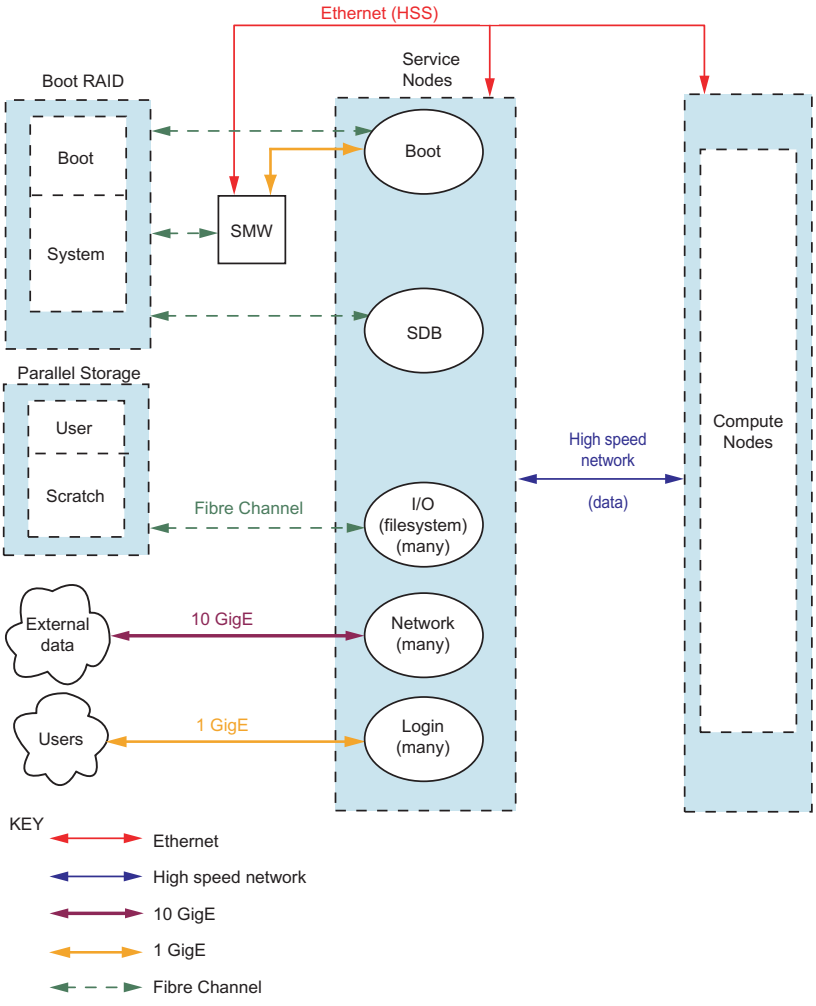
- SLES 11 and Linux documentation — See http://www.novell.com/linux
- Data Direct Networks documentation — See http://www.ddn.com/support/product-downloads-and-documentation
- NetApp, Inc. Engenio® storage system documentation — See http://www.netapp.com/us/products/storage-systems
- MySQL™ documentation — See http://www.mysql.com/documentation
- Lustre File System documentation — See https://wiki.hpdd.intel.com/display/PUB/Documentation
- Batch system documentation:

PBS Professional:	Altair Engineering, Inc.	http://www.pbsworks.com
Moab and TORQUE:	Adaptive Computing Enterprises Inc.	http://www.adaptivecomputing.com
Platform LSF:	Platform Computing Corporation	http://www.platform.com

Introducing System Components [2]

Cray systems separate calculation and monitoring functions. Figure 1 shows the components of a Cray system that an administrator manages.

Figure 1. Administrative Components of a Cray System



A Cray system contains operational components plus storage:

- The System Management Workstation (SMW) is the single point of control for system administration. (For additional information about the SMW, see System Management Workstation (SMW) on page 35.)

  Note: For a system configured for SMW high availability (HA) with the SMW failover feature, there are two SMWs in an active-passive failover configuration. A virtual host name provides access to the active SMW, so that there is still a single point of control. For more information, see *Installing, Configuring, and Managing SMW Failover on the Cray XC30 System* (S–0044).

- The Hardware Supervisory System (HSS) monitors the system and handles component failures. The HSS is an integrated system of hardware and software that monitors components, manages hardware and software failures, controls system startup and shutdown, manages the system interconnection network, and maintains system states. (For additional information about HSS, see Hardware Supervisory System (HSS) on page 45.)

- The Cray Linux Environment (CLE) operating system is the operating system for Cray systems. (For additional information about CLE, see Cray Linux Environment (CLE) on page 36.)

- Service nodes perform the management functions that enable the computations to occur. (For additional information about service nodes, see Service Nodes on page 37.)

- Compute nodes are primarily dedicated to computation. (For additional information about compute nodes, see Compute Nodes on page 42.)

- RAID is partitioned for a variety of storage functions such as boot RAID, database storage, and parallel and user-file system storage. (For additional information about RAID, see Boot Root File System on page 36 and Storage on page 52.)

A Cray system has six network components:

- The 10-GigE network is a high-speed Ethernet pipe that provides external NFS access. It connects to the network nodes and is specifically configured to transfer large amounts of data in and out of the system.
- Users access a 1-GigE network server connection to the login nodes. Logins are distributed among the login nodes by a load-leveling service through the Domain Name Service (DNS) that directs them to the least loaded login node.
- Fibre Channel networks connect storage to the system components.
- The RAID controllers connect to the SMW through the HSS network. This storage sends log messages to the SMW when a failure affects the ability of the disk farm to reliably store and retrieve data.
- The *system interconnection network* includes custom Cray components that provide high-bandwidth, low-latency communication between all the service nodes and compute nodes in the system. The system interconnection network is often referred to as the *high-speed network (HSN)*.
- The HSS network performs the reliability, accessibility, and serviceability functions. The HSS consists of an internet protocol (IP) address and associated control platforms that monitor all nodes.

2.1 System Management Workstation (SMW)

The SMW is the administrator's console for managing a Cray system. The SMW is a server that runs a combination of the SUSE Linux Enterprise Server version 11 operating system with a Service Pack, Cray developed software, and third-party software. The SMW is also a single point of control for the HSS. The HSS data is stored on an internal hard drive of the SMW. For more information about the HSS, see Hardware Supervisory System (HSS) on page 45. For information about installing the SMW release software, see *Installing Cray System Management Workstation (SMW) Software* (S–2480).

Note: For a system configured for SMW HA with the SMW failover feature, see *Installing, Configuring, and Managing SMW Failover on the Cray XC30 System* (S-0044).

You log on to an SMW window on the console to perform SMW functions. From the SMW, you can log on to a disk controller or use a web-browser-based interface from the SMW to configure a RAID controller or Fibre Channel switch. You can log on to the boot node from the SMW as well. From the SMW, you cannot log on directly (`ssh`) to any service node except the boot node.

Most system logs are collected and stored on the SMW. The SMW plays no role in computation after the system is booted. From the SMW, you can initiate the boot process, access the database that keeps track of system hardware, and perform standard administrative tasks.

2.2 Cray Linux Environment (CLE)

CLE is the operating system for Cray systems. CLE is the Cray customized version of the SLES 11 SP3 operating system with a Linux 3.0.93 kernel. This full-featured operating system runs on the Cray service nodes. The Cray compute nodes run a kernel developed to provide support for application execution without the overhead of a full operating-system image. In the compute node root runtime environment (CNRTE), compute nodes have access to the service node shared root (via `chroot`) such that compute nodes can access the full features of a Linux environment.

CLE commands enable administrators to perform administrative functions on the service nodes to control processing. The majority of CLE commands are launched from the boot node, making the boot node the focal point for CLE administration.

For a complete list of Cray developed CLE administrator commands, see Appendix A, SMW and CLE System Administration Commands on page 385.

2.3 Boot Root File System

The boot node has its own root file system, `bootroot`, which is created on the boot RAID during installation. You install and configure the boot RAID from the SMW before you boot the boot node. The boot node mounts the `bootroot` from the boot RAID.

2.4 Shared Root File System

A Cray system has a root file system that is distributed as a read-only shared file system among all the service nodes except the boot node. Each service node has the same directory structure, which is made up of a set of symbolic links to the shared-root file system. For most files, only one version of the file exists on the system, so if you modify the single copy, it affects all service nodes. This makes the administration process similar to that of a single system.

You manage the shared-root file system from the boot node through the `xtopview` command (see Managing System Configuration with the `xtopview` Tool on page 133).

If you need unique files on a specific node or class of nodes (that is, nodes of a certain type), you can set up a modified directory structure. This process, called *specialization*, creates a new directory hierarchy that overlays the existing root directory on the specified nodes and contains symbolic links that point to the unique files. For information about the shared root and file specialization, see Configuring the Shared-root File System on Service Nodes on page 127.

2.5 Service Nodes

Service nodes can be specialized and categorized by the services that run on them.

Service nodes run the CLE operating system. The administrator commands for these nodes are standard Linux commands and Cray system-specific commands.

You log on to the boot node through the SMW console, then from the boot node you can log on to the other service nodes.

Service nodes perform the functions needed to support users, administrators, and applications running on compute nodes. As the system administrator, you define service node classes by the service they perform. Configuration information in the service database on the SDB node determines the functions of the other nodes and services, such as where a batch subsystem runs. In small configurations, some services can be combined on the same node: for example, the `sdb` and `syslog` services can both run on the same node.

You can start services system-wide or on specific nodes. You can start services during the boot process or later on specific nodes of a running system. How you start a service depends on the type of service.

Service nodes, unlike compute nodes, are generally equipped with Peripheral Component Interconnect (PCI) protocol card slots to support external devices.

Service nodes run a full-featured version of the Linux operating system. Service node kernels are configured to enable Non-Uniform Memory Access (NUMA), which minimizes traffic between sockets by using socket-local memory whenever possible.

System management tools are a combination of Linux commands and Cray system commands that are analogous to standard Linux commands but operate on more than one node. For more information about Cray system commands, see Appendix A, SMW and CLE System Administration Commands on page 385. After the system is up, you can access any service node from any other service node, provided you have the correct permissions.

2.5.1 Boot Node

Use the boot node to manage files, add users, and mount and export the shared-root file system to the rest of the service nodes. These shared-root files are mounted from the boot node as read-only.

You can configure two boot nodes per system or per partition, one primary and one for backup (secondary). The two boot nodes must be located on different blades. When the primary boot node is booted, the backup boot node also begins to boot. But the backup boot node boot process is suspended until a primary boot-node failure event is detected. For information about configuring boot-node failover, see Configuring Boot-node Failover on page 180.

2.5.2 Service Database (SDB) Node

The SDB node hosts the service database (SDB), which is a MySQL database that resides on a separate file system on the boot RAID. The SDB is accessible to every service node (see Changing the Service Database (SDB) on page 191). The SDB provides a central location for storing information so that it does not need to be stored on each node. You can access the SDB from any service node after the system is booted, provided you have the correct authorizations.

The SDB stores the following information:

- Global state information of compute processors. This information is used by the Application Level Placement Scheduler (ALPS), which allocates compute processing elements for compute nodes running CNL. For more information about ALPS, see ALPS for Compute Nodes on page 41.
- System configuration tables that list and describe processor attribute and service information.

The SDB node is the second node that is started during the boot process.

You can configure two SDB nodes per system or per partition, one primary and one for backup (secondary). The two SDB nodes must be located on different system blades. For more information, see Configuring SDB Node Failover on page 184.

2.5.3 Syslog Node

By default, the boot node forwards syslog traffic from the service nodes to the SMW for storage in log files. An optional syslog node may be specified (see the `CLEinstall.conf`(5) man page); however, this service node must be provisioned and configured to be able to reach the SMW directly over an attached Ethernet link.

2.5.4 Login Nodes

Users log on to a login node, which is the single point of control for applications that run on the compute nodes. Users do not log on to the compute nodes.

You can use the Linux `lbnamed` load balancer software provided to distribute user logins across login nodes (see Configuring the Load Balancer on page 159). The number of login nodes depends upon the installation and user requirements. For typical interactive usage, a single login node handles 20 to 30 batch users or 20 to 40 interactive users with double this number of user processes.

Caution: Login nodes, as well as other service nodes, do not have swap space. If users consume too many resources, Cray service nodes can run out of memory. When an out of memory condition occurs, the node can become unstable or may crash. System administrators should take steps to manage system resources on service nodes. For example, resource limits can be configured using the `pam_limits` module and the `/etc/security/limits.conf` file. For more information, see the `limits.conf`(5) man page.

2.5.5 Network Nodes

Network nodes connect to the external network with a 10-GigE card. These nodes are designed for high-speed data transfer.

2.5.6 I/O Nodes

I/O nodes host the Lustre file system; see Lustre File System on page 40. The I/O nodes connect to the RAID subsystems that contain the Lustre file system. Two I/O nodes connect to each RAID device for resiliency; each I/O node has full accessibility to all storage on the connected RAID device. Cray provides support for RAID subsystems from two different vendors, Data Direct Networks™ (DDN) and NetApp, Inc.

Cray Data Virtualization Service (Cray DVS) servers run on an I/O node; see Cray Data Virtualization Service (Cray DVS) on page 40. DVS servers cannot run on the same I/O nodes as Lustre servers. On I/O nodes, DVS servers act as external file system clients. DVS will project the external file systems to service and compute node clients within the system.

2.5.7 Services on the Service Nodes

Service nodes provide the services described in this section.

2.5.7.1 Resiliency Communication Agent (RCA)

The RCA is the message path between the CLE operating system and the HSS. The RCA runs on all service nodes and CNL compute nodes.

The `service_config` table of the SDB maintains a list of services that RCA starts. For the services listed in the `service_config` table, the RCA daemon (`rcad_svcs`) starts and restarts all services that must run on a node. You can determine or modify services available through the SDB `service_config` table by using the `xtservconfig` command.

Note: Services can also be started manually or automatically by using standard Linux mechanisms (see Adding and Starting a Service Using Standard Linux Mechanisms on page 226).

The SDB `serv_cmd` table stores information about each service, such as, service type, service instance, heartbeat interval, and restart policy.

The configuration file for service nodes is `/etc/opt/cray/rca/rcad_svcs.service.conf`. By default, this configuration file starts the `rca_dispatcher` and the failover manager.

The RCA consists of a kernel-mode driver and a user-mode daemon on CLE. The RCA driver, `rca.ko`, runs as a kernel-loadable module for the service partition. On CNL compute nodes, the RCA operates through system calls and communicates with the HSS to track the heartbeats (see Blade Controllers and Cabinet Controllers on page 46) of any programs that have registered with it and to handle event traffic between the HSS and the applications that register to receive events. The RCA driver starts as part of the kernel boot, and the RCA daemon starts as part of the initialization scripts.

2.5.7.2 Lustre File System

Cray systems running CLE support the Lustre file system, which provides a high-performance, highly scalable, POSIX-compliant shared file system. You can configure Lustre file systems to operate in the most efficient manner for the I/O needs of applications, ranging from a single metadata server (MDS) and object storage target (OST) to a single MDS with up to 128 OSTs. User directories and files are shared and are globally visible from all compute and service node Lustre clients.

For more information, see *Managing Lustre for the Cray Linux Environment (CLE)* (S–0010) and *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

2.5.7.3 Cray Data Virtualization Service (Cray DVS)

The Cray Data Virtualization Service (Cray DVS) is a parallel I/O forwarding service that provides for transparent use of multiple file systems on Cray systems with close-to-open coherence, much like NFS.

For additional information, see the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) and *Introduction to Cray Data Virtualization Service* (S–0005).

2.5.7.4 ALPS for Compute Nodes

For compute nodes running CNL, the Application Level Placement Scheduler (ALPS) is provided. ALPS provides application placement, launch, and management functionality and cooperates with third-party batch systems for application scheduling. The third-party batch system (such as PBS Professional, Moab, TORQUE, or Platform LSF) makes the policy and scheduling decisions, and ALPS provides a mechanism to place and launch the applications contained within batch jobs. ALPS also supports placement and launch functionality for interactive applications.

An Extensible Markup Language (XML) interface is provided by ALPS for communication with third-party batch systems. This interface is available through use of the `apbasil` client. ALPS uses application resource reservations to guarantee resource availability to batch system schedulers.

The ALPS application placement and launch functionality is provided for applications executing on compute nodes only; ALPS does not provide placement and launch functionality for service nodes.

Note: Only one application can be placed per node; two different executables cannot be run on the same node at the same time.

ALPS is automatically loaded as part of the CNL environment when booting CNL. The RCA starts the ALPS `apinit` daemon on the compute nodes.

When a job is running on CNL compute nodes, the `aprun` process (see Job Launch Commands on page 44) interacts with ALPS to keep track of the processors that the job uses.

For more information about ALPS, see Chapter 8, Using the Application Level Placement Scheduler (ALPS) on page 261.

2.5.7.5 Cluster Compatibility Mode

Cluster Compatibility Mode (CCM) provides the services needed to run most cluster-based independent software vendor (ISVs) applications "out of the box." CCM is tightly coupled to the workload management system. It enables users to execute cluster applications alongside workload-managed jobs running in a traditional MPP batch or interactive queue. Support for dynamic shared objects and expanded services on CNL compute nodes, using the compute node root runtime environment (CNRTE), provide the services to compute nodes within the cluster queue. Essentially, CCM uses the batch system to logically designate part of the Cray system as an emulated cluster for the duration of the job. For more information about CCM, see Chapter 10, Dynamic Shared Objects and Cluster Compatibility Mode in CLE on page 311.

2.5.7.6 Repurposing CNL Compute Nodes as Service Nodes

Some services on Cray systems have resource requirements or limitations (for example, memory, processing power or response time) that you can address by configuring a dedicated service node, such as a Cray Data Virtualization Service (Cray DVS) node or a batch system management (MOM) node. On Cray systems, service I/O node hardware (on a service blade) is equipped with Peripheral Component Interconnect (PCI) protocol card slots to support external devices. Compute node hardware (on a compute blade) does not have PCI slots. For services that do not require external connectivity, you can configure the service to run on a single, dedicated compute node and avoid using traditional service I/O node hardware.

When you configure a node on a compute blade to boot a service node image and perform a service node role, that node is referred to as a *repurposed compute node*.

For additional information, see the section on repurposing compute nodes in *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

2.5.7.7 IP Implementation

Ethernet interfaces handle IP connectivity to external components. Both IPv4 and IPv6 are supported; IPv4 is the default.

Note: The IPv6 capability is limited to the Ethernet interfaces and `localhost`. Therefore, IPv6 connectivity is limited to service nodes that have Ethernet cards installed. Routing of IPv6 traffic between service nodes across the HSN is not supported.

2.6 Compute Nodes

Cray XC30 system compute nodes run the CNL compute node operating system. CNL is a lightweight compute node operating system. It includes a run-time environment based on the SLES 11 SP3 distribution, with a Linux 3.0.93 kernel and with Cray specific modifications. Device drivers for hardware not supported on Cray systems were eliminated from the kernel. CNL features scalability; only the features required to run high-performance computing applications are available on CNL compute nodes. Other features and services are available from service nodes. Cray has configured and tuned the kernel to minimize processing delays caused by inefficient synchronization. CNL compute node kernels are configured to enable Non-Uniform Memory Access (NUMA), which minimizes traffic between sockets by using socket-local memory whenever possible. CNL also includes a set of supported system calls and standard networking.

Cray XC30 systems also support several accelerators: NVIDIA GPGPU (General Purpose Graphics Processing Unit) processors, NVIDIA Telsa K40s and K20X processors, and Inte®l Xeon Phi™ coprocessors.

Several libraries and compilers are linked at the user level to support I/O and communication service. Cray, PGI, PathScale, and the GNU Compiler Collection (GCC) C, C++, and Intel compilers are supported. For information about using modulefiles and configuring the default programming environment, see About Modules and Modulefiles on page 122 and Configuring the Default Programming Environment (PE) on page 124. For information about the libraries that Cray systems host, see the *Cray Application Developer's Environment User's Guide* (S–2396).

The Resiliency Communication Agent (RCA) daemon, `rcad-svcs`, handles node services (see Services on the Service Nodes on page 39).

The Application Level Placement Scheduler (ALPS), handles application launch, monitoring, and signaling and coordinates batch job processing with third-party batch systems. If you are running ALPS, use the `xtnodestat` command to report job information.

The following user-level BusyBox commands are functional on CNL compute nodes: `ash`, `busybox`, `cat`, `chmod`, `chown`, `cp`, `cpio`, `free`, `grep`, `gunzip`, `kill`, `killall`, `ln`, `ls`, `mkdir`, `mktemp`, `more`, `ps`, `rm`, `sh`, `tail`, `test`, `vi`, and `zcat`. For information about supported command options, see the `busybox`(1) man page.

The following administrator-level `busybox` commands and associated options are functional on CNL compute nodes:

- `dmesg -c -n -s`
- `fuser -m -k -s -4 -6 -SIGNAL`
- `logger -s -t -p`
- `mount -a -f -n -o -r -t -w`
- `ping -c -s -q`
- `sysctl -n -w -p -a -A`
- `umount -a -n -r -l -f -D`

A compute-node failure affects only the job running on that node; the rest of the system continues running.

The `CLEinstall` program creates `/var/opt/cray/install/shell_bootimage_LABEL.sh` which uses the `xtclone` and `xtpackage` utilities on the SMW. Use these commands to set up boot images. You can boot CNL on compute nodes. For more information, see Preparing a Service Node and Compute Node Boot Image on page 63, the `xtclone`(8), `xtpackage`(8), and `xtnodestat`(8) man pages, and the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

2.7 Job Launch Commands

Users run applications from a login node and use the `aprun` command to launch CNL applications. The `aprun` command provides options for automatic and manual application placement. With automatic job placement, `aprun` distributes the application instances on the number of processors requested, using all of the available nodes.

With manual job placement, users can control the selection of the compute nodes on which to run their applications. Users select nodes on the basis of desired characteristics (*node attributes*), allowing a placement scheduler to schedule jobs based on the node attributes. To provide the application launcher with a list of nodes that have a particular set of characteristics (attributes), the user invokes the `cnselect` command to specify node-selection criteria. The `cnselect` script uses these selection criteria to query the table of node attributes in the SDB; then it returns a node list to the user based on the results of the query. For an application to be run on CNL compute nodes, the nodes satisfying the requested node attributes are passed by the `aprun` utility to the ALPS placement scheduler as the set of nodes from which to make an allocation. For detailed information about ALPS, see Chapter 8, Using the Application Level Placement Scheduler (ALPS) on page 261.

For more information about the `aprun` and `cnselect` commands, see the `aprun`(1) and `cnselect`(8) man pages.

2.8 Node Health Checker (NHC)

NHC is automatically invoked by ALPS upon the termination of an application. ALPS passes a list of CNL compute nodes associated with the terminated application to NHC. NHC performs specified tests to determine if compute nodes allocated to the application are healthy enough to support running subsequent applications. If not, it removes any compute nodes incapable of running an application from the resource pool. The CLE installation and upgrade processes automatically install and enable NHC software; there is no need for you to change any installation configuration parameters or issue any commands. To configure NHC tests and to optionally configure NHC to use the secure sockets layer (SSL) protocol, see Configuring Node Health Checker (NHC) on page 160.

2.9 Comprehensive System Accounting (CSA)

Comprehensive System Accounting (CSA) is open-source software that includes changes to the Linux kernel so that CSA can collect more types of system resource usage data than under standard Fourth Berkeley Software Distribution (BSD) process accounting. CSA software also contains interfaces for the Linux process aggregates (`paggs`) and jobs software packages. The CSA software package includes accounting utilities that perform standard types of system accounting processing on

the CSA generated accounting files. CSA, with Cray modifications, is included with CLE and runs on login nodes and CNL compute nodes only. For more information, see Chapter 9, Using Comprehensive System Accounting on page 297.

2.10 Optional Workload-management (Batch) System Software Products

For information about optional batch systems software products for Cray systems, see the following websites.

PBS Professional:	Altair Engineering, Inc.	http://www.pbsworks.com
Moab and TORQUE:	Cluster Resources, Inc.	http://www.clusterresources.com
Platform LSF:	International Business Machines Corporation	http://www.platform.com

2.11 Hardware Supervisory System (HSS)

The HSS is an integrated system of hardware and software that monitors the hardware components of the system and proactively manages the health of the system. The HSS communicates with nodes and with the management processors over an internal (private) Ethernet network that operates independently of the system interconnection network. The HSS data is stored on an internal hard drive of the SMW.

For a complete list of Cray developed HSS commands, see Appendix A, SMW and CLE System Administration Commands on page 385.

The HSS includes the following components:

- The HSS network (see HSS Network on page 46).
- The HSS interface (see HSS Interface on page 46).
- Blade and cabinet control processors (see Blade Controllers and Cabinet Controllers on page 46).
- Network Time Protocol (NTP) server (see NTP Server on page 47).
- Event router (see Event Router on page 47).
- HSS managers (see HSS Managers on page 47).

- `xtdiscover` command (see Automatically Discover and Configure Cray System Hardware on page 50).
- Various logs (see Event Logs on page 51, Boot Logs on page 51, Dump Logs on page 51).

2.11.1 HSS Network

The SMW, with its HSS Ethernet network, performs reliability, accessibility, and serviceability tasks. The HSS commands monitor and control the physical aspects of the system.

The SMW manages the HSS network. A series of Ethernet switches connects the SMW to all the cabinets in the system.

2.11.2 HSS Interface

The HSS has a command-line interface to manage and monitor your system. You can use the command-line interface to manage your Cray system from the SMW. For usage information, see Chapter 3, Managing the System on page 63 and Chapter 4, Monitoring System Activity on page 97. For a list of all HSS system administration commands, see Appendix A, SMW and CLE System Administration Commands on page 385.

2.11.3 Blade Controllers and Cabinet Controllers

A blade control processor *(blade controller)* is hierarchically the lowest component of the monitoring system. One blade controller resides on each compute blade and service blade, monitoring only the nodes and ASICs. It provides access to status and control registers for the components of the blade. The blade controller also monitors the general health of components, including items such as voltages, temperature, and other failure indicators. A version of Linux optimized for embedded controllers runs on each blade controller.

Note: In some contexts, the blade controller is referred to as a *slot*.

On Cray XC30 systems, the blade controller is referred to as the *BC*.

Each cabinet has a cabinet control processor (*cabinet controller*) that monitors and controls the cabinet power and cooling equipment and communicates with all the blade controllers in the cabinet. It sends a periodic heartbeat to the SMW to indicate cabinet health.

The cabinet controller connects to the chassis controller and in turn the chassis controller connects to the blade controllers (via the backplane) on each blade by Ethernet cable and routes HSS data to and from the SMW. The cabinet controller runs embedded Linux.

The monitoring system operates by periodic heartbeats. Processes send heartbeats within a time interval. If the interval is exceeded, the system monitor generates a fault event that is sent to the state manager. The fault is recorded in the event log, and the state manager (see State Manager on page 48) sets an alert flag for the component (blade controller or cabinet controller) that spawned it.

The cabinet and blade controllers use `ntpclient` to keep accurate time with the SMW.

You can dynamically configure the cabinet controller system daemon and the blade controller system daemon with the `xtdaemonconfig --`*daemon_name* command (see the `xtdaemonconfig`(8) man page for detailed information).

Note: There is no NV write protection feature on the cabinet and blade controllers; you should not assume the write protection functionality on the cabinet controller front panel display will protect the NV memory on the cabinet and blade controllers.

2.11.4 NTP Server

The SMW workstation is the primary NTP server for the Cray system. The blade controllers use the HSS network to update themselves according to the NTP protocol. To change the NTP server, see Configuring the SMW to Synchronize to a Site NTP Server on page 225.

2.11.5 Event Router

HSS functions are event-driven. The event router daemon, `erd`, is the root of the HSS. It is a system daemon that runs on the SMW, cabinet controllers, and blade controllers. The SMW runs a separate thread for each cabinet controller. The cabinet controller runs a separate thread for each blade controller. HSS managers subscribe to events and inject events into the HSS system by using the services of the `erd`. (For descriptions of HSS managers, see HSS Managers on page 47) The event router starts as each of the devices (SMW, cabinet controller, blade controller) are started.

When the event router on the SMW receives an event from either a connected agent or from another event router in the hierarchy, the event is logged and then processed. The `xtcli` commands, which are primary HSS control commands, also access the event router to pass information to the managers.

The `xtconsumer` command (see Monitoring Events on page 106) monitors the `erd`. The `xtconsole` command (see Monitoring Node Console Messages on page 107) operates a shell window that displays all node console messages.

2.11.6 HSS Managers

HSS managers are located in `/opt/cray/hss/default/etc`. They report to the event router and get information from it.

The HSS managers are started by running the `/etc/init.d/rsms start` command.

You can configure HSS daemons dynamically by executing the `xtdaemonconfig` command. For a list of the HSS daemons, see the `xtdaemonconfig`(8) man page. This section highlights the following key HSS daemons:

- state manager
- boot manager
- system environmental data collections (SEDC) manager
- NID manager

2.11.6.1 State Manager

Every component has a state at all times. The state manager, `state_manager`, runs on the SMW and uses a relational database (also referred to as the *HSS database*) to read and write the system state. The state manager keeps the database up to date with the current state of components and retrieves component information from the database when needed. The state manager uses the Lightweight Log Manager (LLM). The log data from state manager is written to `/var/opt/cray/log/sm-`*yyyymmdd*. The default setting for state manager is to enable LLM logging. In addition, the dynamic system state persists between boots. The state manager performs the following functions:

- Updates and maintains component state information (see Appendix B, System States on page 393)
- Monitors events to update component states
- Detects and handles state notification upon failure
- Provides state and configuration information to HSS applications so that they do not interfere with other applications working on the same component

The state manager listens to the `erd`, records changes of states, and shares those states with other daemons.

2.11.6.2 Boot Manager

The boot manager, `bootmanager`, runs on the SMW. It controls the acts of placing kernel data into node memories and requesting that they begin booting.

During the boot process, the state manager provides state information that allows the nodes to be locked for booting. After the nodes boot, the state manager removes the locks and notifies the boot manager. The boot manager logging facility includes a timestamp on log messages.

2.11.6.3 System Environmental Data Collections (SEDC) Manager

The System Environment Data Collections (SEDC) manager, `sedc_manager`, monitors the system's health and records the environmental data and status of hardware components such as power supplies, processors, temperature, and fans. SEDC can be set to run at all times or only when a client is listening. The SEDC configuration file provided by Cray has automatic data collection set as the default action.

The SEDC configuration file (`/opt/cray/hss/default/etc/sedc_srv.ini` by default) configures the SEDC server. In this file, you can also create sets of different configurations as groups so that the blade and cabinet controller daemons can scan components at different frequencies. The `sedc_manager` sends out the scanning configuration for specific groups to the cabinet and blade controllers and records the incoming data by group. For information about configuring the SEDC manager, see *Using and Configuring System Environment Data Collections (SEDC)* (S–2491) and the `sedc_manager`(8) man page.

To view System Environment Data Collections (SEDC) scan data, use the `xtsedcviewer` command-line interface. This utility allows you to view the server configurations (groups) as well as the SEDC scan data from blade and cabinet controllers. For information about viewing SEDC server configuration and the SEDC scan data, see *Using and Configuring System Environment Data Collections (SEDC)* (S–2491) and the `xtsedcviewer`(8) man page.

2.11.6.4 NID Manager

The NID (node ID) manager, `nid_mgr`, runs on the SMW and provides a NID mapping service for the rest of the HSS environment.

Along with the ability to assign NIDs automatically, the `nid_mgr` supports a mechanism that allows an administrator to control the NID assignment; this is useful for handling unique configurations. Administrator-controlled NID assignment is accomplished through the `nids.ini` NID assignment file.

Caution: The `nids.ini` file can have a major impact on the functionality of a Cray system and should only be used or modified at the recommendation of Cray support personnel. Setting up this file incorrectly could make the Cray system unroutable.

Typically, after a NID mapping is defined for a system, this mapping is used until some major event occurs, such as a hardware configuration change (see Updating the System Configuration After a Blade Change on page 214). This may require the NID mapping to change, depending on the nature of the configuration change. Adding additional cabinets to the ends of rows does not typically result in a new mapping. Adding additional rows most likely does result in a new mapping. If the configuration change is such that the topology class of the system is changed, this will require a new NID mapping. Otherwise, the NID mapping remains static.

The `nid_mgr` generates a list of mappings between the physical location and Network Interface Controller ID (NIC ID) and distributes this information to the blade controllers. Because the operating system always uses node IDs (NIDs), the HSS converts these to NIC IDs when sending them onto the HSS network and back to NIDs when forwarding events from the HSS network to a node.

For more information about node IDs, see Identifying Components on page 53.

2.11.7 Automatically Discover and Configure Cray System Hardware

The `xtdiscover` command automatically discovers the hardware components on a Cray system and creates entries in the system database to reflect the current hardware configuration. The `xtdiscover status` command can correctly identify missing or nonresponsive cabinets, empty or nonfunctioning slots, the blade type (service or compute) in each slot, and the CPU type and other attributes of each node in the system. The `xtdiscover` command and the state manager ensure that the system status represents the real state of the hardware. When it has finished, you can use the `xtcli` command to display the current configuration. No previous configuration of the system is required; the hardware is discovered and made available, and you can modify the components after `xtdiscover` has finished creating entries in the system database.

The `xtdiscover` interface steps a system administrator through the discovery process. The `xtdiscover.ini` file allows you to predefine values such as topology class, cabinet layout, and so on. A template `xtdiscover.ini` file is installed with the SMW software. The default location of the file is `/opt/cray/hss/default/etc/xtdiscover.ini`.

Note: When `xtdiscover` creates a default partition, it uses `c0-0c0s0n1` as the default for the boot node and `c0-0c0s1n1` as the default SDB node.

The `xtdiscover` command does not use or configure the Cray High Speed Network (HSN). The HSN configuration is done when booting the system with the `xtbootsys` command.

If there are changes to the system hardware, such as adding a new cabinet or removing a blade and replacing it with a blade of a different type (for example, a service blade that is replaced with a compute blade), then `xtdiscover` must be executed again, and it will perform an incremental discovery of the hardware changes without disturbing the rest of the system.

For more information, see the `xtdiscover`(8) man page.

2.11.8 Cray System Network Routing Utility

Use the `rtr` command to perform a variety of routing-related tasks. The `rtr` command is also invoked as part of the `xtbootsys` process. For more information, see the `rtr`(8) man page.

2.11.9 Log Files

For more information about examining log files, see Managing Log Files Using CLE and HSS Commands on page 98.

2.11.9.1 Event Logs

The event router records events to the event log in the `/var/opt/cray/log/event-`*yyyymmdd* file. When the log grows beyond a reasonable size, it turns over and its contents are stored in a numbered file in the directory.

2.11.9.2 Boot Logs

The `/var/opt/cray/log/`*session-id* directory is a repository for files collected by commands such as `xtbootsys`, `xtconsole`, `xtconsumer`, and `xtnetwatch` for the currently booted session. To determine the current *sessionid*, see the `xtsession`(8) man page.

Note: A symbolic link will be created from `/var/opt/cray/log/`*partition*`-current` to the currently booted session directory.

2.11.9.3 Dump Logs

The `/var/opt/cray/dump` directory is a repository for files collected by the `xtdumpsys` command. It contains time-stamped dump files.

For more information about examining log files, see Managing Log Files Using CLE and HSS Commands on page 98.

2.12 SEC Software for Log Monitoring and Event Processing

The simple event correlator (SEC) is released under the GNU Public License (GPL) v2. As described at http://simple-evcorr.sourceforge.net/, SEC is "... an event correlation tool for advanced event processing which can be harnessed for event log monitoring, for network and security management, for fraud detection, and for any other task which involves event correlation. *Event correlation* is a procedure where a stream of events is processed, in order to detect (and act on) certain event groups that occur within predefined time windows. Unlike many other event correlation products which are heavyweight solutions, SEC is a lightweight and platform-independent event correlator which runs as a single process. The user can start it as a daemon, employ it in shell pipelines, execute it interactively in a terminal, run many SEC processes simultaneously for different tasks, and use it in a wide variety of other ways."

A simplified description of SEC is that it parses every line being appended to system log files, watches for specific strings to show up that represent significant events occurring in the system, and sends out E-mail notification that the event has occurred.

For information about optionally using SEC for your Cray system, see *Configuring SEC Software for a Cray XC, Cray XE, or Cray XK System* (S–2542).

2.13 Storage

All Cray XC30 systems require RAID storage. RAID storage consists of one or more physical RAID subsystems; a RAID subsystem is defined as a pair of disk controllers and all disk modules that connect to the controllers.

Functionally, there are two types of RAID subsystems: system RAID (also referred to as *boot RAID*) and file system RAID. The system RAID stores the boot image and system files and is also partitioned for database functionality, while the file system RAID stores user files.

File system RAID subsystems use the Lustre file system. Lustre offers high performance scalable parallel I/O capabilities, POSIX semantics, and scalable metadata access. For more information on Lustre file system configuration, see *Managing Lustre for the Cray Linux Environment (CLE)* (S–0010).

Cray offers RAID subsystems from two vendors: DataDirect Network (DDN) and NetApp. All DDN RAID subsystems function as dedicated file system RAID, while NetApp RAID subsystems can function as dedicated file system RAID, a dedicated system RAID, or a combination of both. Different controller models support Fibre Channel (FC), Serial ATA (SATA), and Serial Attached SCSI (SAS) disk options. In addition to vendor solutions for file system RAID, Cray offers an integrated file system, software and storage product, the Sonexion.

RAID devices are commonly configured with zoning so that only appropriate service nodes see the disk devices (LUNs) for the services that will be provided by each node; this is done in order to reduce the possibility of accidental or unauthorized access to LUNs.

Caution: Because the system RAID disk is accessible from the SMW, the service database (SDB) node, the boot node, and backup nodes, it is important that you **never** mount the same file system in more than one place at the same time. If you do so, the Linux operating system will corrupt the file system.

For more information about configuring RAID, see the documentation for your site's particular RAID setup.

2.14 Other Administrative Information

This section contains additional information that is helpful for the administrator.

2.14.1 Identifying Components

System components (nodes, blades, chassis, cabinets, etc.) are named and located by node ID, IP address, physical ID, or class number. Some naming conventions are specific to CLE.

2.14.1.1 Physical ID

The physical ID identifies the cabinet's location on the floor and the component's location in the cabinet as seen by the HSS.

The table below shows the physical ID naming conventions. Descriptions assume that you are standing in front of the system cabinets.

Table 1. Physical ID Naming Conventions

Component	Format	Description
SMW	`s0`, `all`	All components attached to the SMW. `xtcli power up s0` powers up all components attached to the SMW.
cabinet	c*X*-*Y*	Compute/service cabinet, cabinet controller hostname. Not used for blower cabinets. For example: `c12-3` is cabinet 12 in row 3.

Component	Format	Description
compute/service cabinet controller HSS microcontroller	c*X*-*Y*m*M*	Compute/Service cabinet controller HSS microcontroller; *M* is 0.
power rectifier module within a cabinet	c*X*-*Y*r*R*	Power rectifier module within a cabinet; *R* is 0 to 63.
cabinet controller (CC) FPGA	c*X*-*Y*f*F*	Cabinet controller (CC) FPGA; *F* is 0.
blower cabinet	b*X*-*Y*	Blower cabinet, cabinet controller hostname (if applicable). *X* is 0 to 63; *Y* is 0 to 15. For example: `b12-3` is blower cabinet 12 in row 3.
blower cabinet controller	b*X*-*Y*m*M*	Blower cabinet, cabinet controller; *M* is 0.
blower within a blower cabinet	b*X*-*Y*b*B*	Blower within a blower cabinet; *B* is 0-5.
chassis	c*X*-*Y*c*C*	Physical unit within cabinet: c*X*-*Y*; c*C* is the chassis number and *C* is 0-2. Chassis are numbered bottom to top. For example: `c0-0c2` is chassis 2 of cabinet `c0-0`.
chassis host controller	c*X*-*Y*c*C*m*M*	Chassis host controller; *M* is 0.
optical connectors	c*X*-*Y*c*C*j*J*	Optical connectors per chassis; there are 40 optical connectors per chassis. *J* is 0-63.
chassis host FPGA	c*X*-*Y*c*C*f*F*	Chassis host FPGA; *F* is 0.
blade or slot	c*X*-*Y*c*C*s*S*	Physical unit within a slot of a chassis c*X*-*Y*c*C*; s*S* is the slot number of the blade and *S* is 0-15. For example: `c0-0c2s4` is slot 4 of chassis 2 of cabinet `c0-0`. For example: `c0-0c2s*` is all slots (0...15) of chassis 2 of cabinet `c0-0`.

Component	Format	Description
optical controller groups	c*X*-*Y*c*C*o*O*	Optical controller groups – controller groups are associated with slots by multiplying controller number by 2 (and optionally adding 1); *O* is 0-7.
individual optical controller	c*X*-*Y*c*C*o*O*x*X*	Individual optical controller within an optical controller group; *X* is 0-4.
L0D FPGA within a base blade	c*X*-*Y*c*C*s*S*f*F*	L0D FPGA within a base blade; *F* is 0.
Aries™ ASIC	c*X*-*Y*c*C*s*S*a*A*	Aries ASIC within a base blade. There is only one Aries ASIC per blade, and all nodes on the blade connect to it. a*A* is the location of the ASIC within the blade and *A* is 0. For example: `c0-1c2s3a0`.
Aries NIC	c*X*-*Y*c*C*s*S*a*A*n*NIC*	NIC (Network Interface Controller) within an Aries ASIC; *NIC* is 0-3. For example: `c0-1c2s3a0n1`
LCB tile row/column	c*X*-*Y*c*C*s*S*a*A*l*RCol*	LCB tile row/column. Row 5 is all processor tiles; all other rows contain only HSN tiles. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
SerDes macro associated with an LCB	c*X*-*Y*c*C*s*S*a*A*m*RCol*	SerDes macro associated with an LCB. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
SerDes macro network processor associated with an LCB	c*X*-*Y*c*C*s*S*a*A*p*RCol*	SerDes macro network processor associated with an LCB. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
Aries ASIC VRM	c*X*-*Y*c*C*s*S*a*A*v*V*	Aries ASIC VRM; *V* is 0.
Processor Daughter Card (PDC) within a base blade	c*X*-*Y*c*C*s*S*p*P*	Processor Daughter Card within a base blade; *P* is 0-3.
quad Processor Daughter Card (QPDC) within a base blade	c*X*-*Y*c*C*s*S*q*Q*	Quad Processor Daughter Card within a base blade; *Q* is 0-1.

Component	Format	Description
general-purpose-accelerator Processor Daughter Card (GPDC) within a base blade	c*X*-*Y*c*C*s*S*k*K*	General-purpose-accelerator Processor Daughter Card (GPDC) within a base blade; *K* is 0-1.
L0C FPGA within a PDC	c*X*-*Y*c*C*s*S*p*P*f*F*	L0C FPGA within a PDC; *F* is 0.
L0C FPGA within a QPDC	c*X*-*Y*c*C*s*S*q*Q*f*F*	L0C FPGA within a Quad PDC; *F* is 0.
L0C FPGA within a GPDC	c*X*-*Y*c*C*s*S*k*K*f*F*	L0C FPGA within a GPDC; *F* is 0.
VRM within a PDC associated with a processor socket	c*X*-*Y*c*C*s*S*p*P*v*V*	VRM within a PDC associated with a processor socket; *V* is 0-1.
SouthBridge chip within a PDC	c*X*-*Y*c*C*s*S*p*P*s*SouthBridge*	SouthBridge chip within a PDC; *SouthBridge* is 0.
SouthBridge chip within a QPDC	c*X*-*Y*c*C*s*S*q*Q*s*SouthBridge*	SouthBridge chip within a Quad PDC; *SouthBridge* is 0-1.
SouthBridge chip within a GPDC	c*X*-*Y*c*C*s*S*k*K*s*SouthBridge*	SouthBridge chip within a GPDC; *SouthBridge* is 0-1.
blade controller HSS microcontroller within a base blade	c*X*-*Y*c*C*s*S*m*M*	Blade controller HSS microcontroller within a base blade (not the blade controller CPU); *M* is 0.
node	c*X*-*Y*c*C*s*S*n*N*	Physical node on a base blade; n*N* is the location of the node and *N* is 0-3. For example: `c0-0c2s4n0` is node 0 on blade 4 of chassis 2 in cabinet `c0-0`. For example: `c0-0c2s4n*` is all nodes on blade 4 of chassis 2 of cabinet `c0-0`.
accelerator	c*X*-*Y*c*C*s*S*n*N*a*A*	Accelerator associated with a node; may be any type of supported accelerator. *A* is 0-7.
processor socket associated with a physical node	c*X*-*Y*c*C*s*S*n*N*s*Socket*	Processor socket associated with a physical node; *Socket* is 0-1.
DIMM associated with a processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*m*M*	DIMM associated with a processor socket; *M* is 0-7.
VDD VRM associated with processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*v*V*	VDD VRM associated with processor socket; *V* is 0.

Component	Format	Description
VDR VRM associated with processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*r*R*	VDR VRM associated with processor socket; *R* is 0.
die within a processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*	Die within a processor socket; *D* is 0-3.
core within a die	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*c*Core*	Core within a die; *Core* is 0-63.
memory controller within a die	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*m*M*	Memory controller within a die; *M* is 0-3.
logical machine (partition)	p#	A partition is a group of components that make up a logical machine. Logical systems are numbered from 0 to the maximum number of logical systems minus one. Because p0 is reserved to refer to the entire machine as a partition a configuration with 31 logical machines would be numbered p1 through p31 (see Logical Machines on page 61) and p0 would need to be deactivated or removed as it would no longer be valid.

2.14.1.2 Node ID (NID)

The node ID (NID) is a decimal numbering of all CLE nodes. NIDs are sequential numberings of the nodes starting in cabinet c0-0. Each additional cabinet continues from the highest value of the previous cabinet; so, cabinet 0 has NIDs 0-191, and cabinet 1 has NIDs 192 - 383, and so on.

A cabinet contains three chassis; chassis 0 is the lower chassis in the cabinet. Each chassis contains sixteen blades and each blade contains four nodes. The lowest numbered NID in the cabinet is in chassis 0 slot 0 (lower left corner); slots (blades) are numbered left to right (slot 0 to slot 15; as you face the front of the cabinet). NID numbering begins in cabinet 0, cage 0, slot 0 with NIDs 0, 1, 2, and 3; NIDs 4, 5, 6, and 7 are in slot 1, and this numbering scheme continues to slot 15 and then moves up to chassis 1 and so on.

Use the xtnid2str command to convert a NID to a physical ID. For information about using the xtnid2str command, see the xtnid2str(8) man page. To convert a physical ID to a NID number, you can use the rtr --system-map command and filter the output. For example:

```
crayadm@smw:~> rtr --system-map | grep c1-0c0s14n3 | awk '{ print $1 }'
251
```

Use the `nid2nic` command to print the *nid*-to-*nic_address* mappings, *nic_address*-to-*nid* mappings, and a specific *physical_location*-to-*nic_address* and *nid* mappings. For information about using the `nid2nic` command, see the `xtnid2str`(8) man page.

2.14.1.3 Extended Node ID (XNID)

An extended node ID (XNID) provides a means of addressing host nodes and their Intel Xeon Phi coprocessors independently even though a host and coprocessor share the same network interface. An XNID provides a handle for common communication interfaces within the system, such as PMI, LNET, TCP/IP, and DVS, to access coprocessors. This direct access permits direct (autonomous) execution of coprocessor-targeted executables.

During the installation of a system with Xeon Phi components, the CLE installer prompts for a base extended node identifier offset value for the system. For example, assume that base is set to `50000`. That number is added to the host NID for a node containing a coprocessor. If the host node is `nid00032`, then the coprocessor is `nid50032`.

2.14.1.4 Class Name

Class names are a CLE construct. The `/etc/opt/cray/sdb/node_classes` file is created as part of the system installation, based on the `node_class*` settings defined in `CLEinstall.conf`. During the boot process, the `service_processor` database table is populated from the `/etc/opt/cray/sdb/node_classes` file, which can be changed if you add or remove nodes (see Changing Nodes and Classes on page 195).

Note: It is important to keep node class settings in `CLEinstall.conf` and `/etc/opt/cray/sdb/node_classes` consistent in order to avoid errors during update or upgrade installations (see the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

The only restriction about how you name the classes is that the class name must be valid (e.g., the name of a directory.) There is no restriction about how many classes you specify; however, you must use the same class names when you invoke the `xtspec` specialization command (see Specializing Files on page 136).

Change the class of a node (see Changing the Class of a Node on page 141) when you change its function, for example, when you have added an additional login node.

The `/etc/opt/cray/sdb/node_classes` file describes the nodes associated with each class.

Example 1. Sample `/etc/opt/cray/sdb/node_classes` file

```
# node:classes
0:service
1:service
8:login
9:service
```

2.14.2 Topology Class

Each Cray system is given a topology class based in the number of cabinets and their cabling. Some commands, such as `xtbounce`, let you specify topology class as an option.

You can see the class value of your system in a number of places, such as `xtcli status` output, `rca-helper -o` command output (`rca-helper` is run from a Cray node), or by using the `xtclass` command from the SMW:

```
smw:~> xtclass
1
```

2.14.3 Persistent `/var` Directory

You can set up a persistent, writable `/var` directory on each service node served with NFS. The boot node has its own root file system and its own `/var` directory; the boot node `/var` is not part of the NFS exported `/snv` file system.

Because the Cray system root file system is read-only, some subdirectories of `/var` are mounted on `tmpfs` (memory) and not on disk. Persistent `/var` retains the contents of `/var` directories between system boots. If `persistent_var=yes` in `CLEinstall.conf`, `CLEinstall` configures the correct values for `VAR_SERVER`, `VAR_PATH`, and `VAR_MOUNT_OPTIONS` in the `/etc/sysconfig/xt` file during installation so the service nodes will NFS mount the proper path at boot time.

Boot scripts and the `xtopview` utility (see Managing System Configuration with the `xtopview` Tool on page 133) respect these configuration values and mount the correct `/var` directory.

For more information, see the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

2.14.4 Default Network IP Addresses

The default IP addresses for network components are described in the *Installing Cray System Management Workstation (SMW) Software* (S–2480).

2.14.5 /etc/hosts Files

The host file on the boot node is for the HSN and external hosts accessible from login and network nodes. The hosts file on the SMW is for the HSS network.

The xtcdr2proc utility takes information from the Resiliency Communication Agent (RCA) to build the /etc/hosts file on the boot node. The /etc/hosts file on the boot node maps IP addresses to node IDs on the system interconnection network (see Identifying Components on page 53). The file can also contain aliases for the physical ID location of the system interconnection network components and class names. The /etc/hosts file is updated or created at boot time and contains the default hostname mappings as well as service and HSS names. The upper octets typically range from 10.128.0.0 to 10.131.255.255. Lower octets for nodes are derived from their NID. The NID is a sequential numbering of nodes from cabinet 0 up.

The /etc/hosts file on the boot node is generated at boot time to include the compute nodes. Also, the installation and upgrade process modifies the /etc/hosts file on the boot root to include compute nodes if they are not included.

The /etc/hosts file on the SMW contains physIDs (physical IDs that map to the physical location of HSS network components), such as the blade and cabinet controllers (see Physical ID on page 53).

The default system IP addresses are shown in the *Installing Cray System Management Workstation (SMW) Software* (S–2480).

The xtdb2etchosts command converts service information in the SDB to an /etc/hosts style file. The resulting /etc/hosts file has lines of the following form, where the first column is the IP address, the second column is the NID, and the third column is the service type and class ID of the node:

```
10.131.255.254 nid12345  boot001
10.131.255.253 nid67890  boot002
10.131.255.252 nid55512  login001
```

The service configuration table (service_config) in the SDB XTAdmin database provides a line for each service IP address of the form, where SERV1 and SERV2 are the service names in the service_config table:

```
1.2.3.1    SERV1
1.2.3.2    SERV2
```

Note: Each time you update or upgrade your CLE software, CLEinstall verifies the content of /etc/opt/cray/sdb/node_classes and modifies /etc/hosts to match the configuration specified in your CLEinstall.conf file. For additional detail about how CLEinstall modifies the /etc/hosts file, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

The xtdb2etchosts command is documented on the xtdb2etchosts(8) man page.

2.14.6 Realm-Specific IP Addressing (RSIP) for Compute Nodes

Realm-Specific IP Addressing (RSIP) allows compute nodes and the service nodes to share the IP addresses configured on the external Gigabit and 10 Gigabit Ethernet interfaces of network nodes. By sharing the external addresses, you may rely on your system's use of private address space and do not need to configure compute nodes with addresses within your site's IP address space. The external hosts see only the external IP addresses of the Cray system.

To configure RSIP for compute nodes, see Configuring Realm-specific IP Addressing (RSIP) on page 209.

2.14.7 Logging Failed Login Attempts

The `cray_pam` module is a Pluggable Authentication Module (PAM) that, when configured, provides information to the user at login time about any failed login attempts since their last successful login. For more information, see Using the `cray_pam` PAM to Log Failed Login Attempts on page 152.

2.14.8 Logical Machines

You can subdivide a single Cray system into two or more logical machines (partitions), which can then be run as independent systems. An operable logical machine has its own compute nodes and service nodes, external network connections, boot node, and SDB node. Each logical machine can be booted and dumped independently of the other logical machines. Once booted, a logical machine appears as a normal Cray system to the users, limited to the set of hardware included for the logical machine.

The HSS is common across all logical machines. Because logical machines apply from the system interconnection network layer and up, the HSS functions continue to behave as a single system for power control, diagnostics, low-level monitoring, and so on.

In addition,

- Cray recommends that you do not configure more than one logical machine per cabinet. That way, if you power down a cabinet, you do not affect more than one logical machine. A logical machine can include more than one cabinet.
- A job is limited to running within a single logical machine.
- Although the theoretical maximum of allowable logical machines per physical Cray system is 31 logical machines (as `p0` is the entire system), you must consider your hardware requirements to determine a practical number of logical machines to configure.
- Because no two logical machines can use the same components, once a system is partitioned into logical machines `p0` is no longer a valid reference and should be removed or deactivated.
- You can run only a single instance of SMW software.
- Boot and routing commands affect only a single logical machine.

To create logical machines, see Creating Logical Machines on page 186.

Managing the System [3]

Important: SCSI device names (`/dev/sd*`) are not guaranteed to be numbered the same from boot to boot. This inconsistency can cause serious system problems following a reboot. When installing CLE, you must use persistent device names for file systems on your Cray system. This does **not** apply to SMW disks. For additional information, see Using Persistent SCSI Device Names on page 258.

3.1 Connecting the SMW to the Console of a Service Node

The `xtcon` command is a console interface for service nodes. When it is executing, the `xtcon` command provides a two-way connection to the console of any running service node.

See the `xtcon`(8) man page for additional information.

3.2 Logging on to the Boot Node

The standard Cray configuration has a gigabit Ethernet connection between the SMW and boot node. You can access the other nodes on the Cray system from the boot node.

Procedure 1. Logging on to the boot node

- From the SMW, log on to the boot node.

```
crayadm@smw:~> ssh boot
crayadm@boot:~>
```

Note: You can open an administrator window on the SMW to access the boot node:

```
crayadm@smw:~> xterm -ls -vb -sb -sl 2049 6&
```

After the window opens, use it to `ssh` to the boot node.

3.3 Preparing a Service Node and Compute Node Boot Image

This section describes how to prepare a Cray service node and compute node boot image.

A *boot image* is an archive containing all the components necessary to boot Cray service nodes and compute nodes. In general, a boot image contains the operating system kernel, ramdisk, and boot parameters used to bring up a node. A single boot image can contain multiple sets of these files to support booting service nodes and compute nodes from the same boot image as well as booting different versions of compute node operating systems. The operating systems supported by a particular boot image are described through load files. A *load file* is simply a manifest of operating system components to include (represented as files) and load address information to provide to the boot loader. Load files should not be edited by the administrator.

Cray system compute and service nodes use a RAM disk for booting. Service nodes and compute nodes use the same `initramfs` format and work space environment. This space is created in `/opt/xt-images/`*machine*`-`*xtrelease*`-`*LABEL*`[-`*partition*`]/`*nodetype*, where *machine* is the Cray hostname, *xtrelease* is the CLE release level, *LABEL* is the system set label in `/etc/sysset.conf`, *partition* describes a system partition and is typically omitted if partitions are not used, and *nodetype* is either `compute` or `service`.

3.3.1 Using `shell_bootimage_`*LABEL*`.sh` to Prepare Boot Images

The `CLEinstall` installation program creates a `/var/opt/cray/install/shell_bootimage_`*LABEL*`.sh` script on the SMW. This script is unique to the system set label you installed, based on settings in the `CLEinstall.conf` and `/etc/sysset.conf` installation configuration files. You can reuse this script to automate some of the steps for creating boot images.

Procedure 2. Preparing a boot image for CNL compute nodes and service nodes

Invoke the `shell_bootimage_`*LABEL*`.sh` script to prepare boot images for the system set with the specified *LABEL*. This script uses `xtclone` and `xtpackage` to prepare the work space in `/opt/xt-images`.

`shell_bootimage_`*`LABEL`*`.sh` accepts the following options:

`-v` Run in verbose mode.

`-T` Do not update the `default` template link.

`-h` Display help message.

`-c` Create and set the boot image for the next boot. The default is to display `xtbootimg` and `xtcli` commands that will generate the boot image. Use the `-c` option to invoke these commands automatically.

`-b` *bootimage*

Specify *bootimage* as the boot image disk device or file name. The default *bootimage* is determined by using values for the system set *LABEL* when `CLEinstall` was executed. Use this option to override the default and manage multiple boot images.

Optionally, this script includes `CNL_*` parameters that you can use to modify the CNL boot image configuration you defined in `CLEinstall.conf`. Edit the script and set the associated parameter to **`y`** to load an optional RPM or change the `/tmp` configuration.

1. Execute `shell_bootimage_`*`LABEL`*`.sh`, where *LABEL* is the system set label specified in `/etc/sysset.conf` for this boot image. For example, if the system set label is *BLUE*, log on to the SMW as root and type:

   ```
   smw:~# /var/opt/cray/install/shell_bootimage_BLUE.sh
   ```

   On completion, the script displays the `xtbootimg` and `xtcli` commands required to build and set the boot image for the next boot. If you specified the `-c` option, the script invokes these commands automatically and you should skip the remaining steps in this procedure.

2. Create a unified boot image for compute and service nodes by using the `xtbootimg` command suggested by the `shell_bootimage_`*`LABEL`*`.sh` script.

   In the following example, replace *bootimage* with the *mountpoint* for `BOOT_IMAGE0` in the system set defined in `/etc/sysset.conf`. Set *bootimage* to either a raw device; for example `/raw0` or a file name; for example `/bootimagedir/bootimage.new`.

Caution: If *bootimage* is a file, verify that the file exists in the same path on both the SMW and the boot root.

Type the following command:

```
smw:~# xtbootimg \
-L /opt/xt-images/machine-xtrelease-LABEL/compute/CNL0.load \
-L /opt/xt-images/machine-xtrelease-LABEL/service/SNL0.load \
-c bootimage
```

a. At the prompt 'Do you want to overwrite', type **y** to overwrite the existing boot image file.

b. If *bootimage* is a file, mount the boot node root file system to `/bootroot0`, copy the boot image file from the SMW to the same directory on the boot root, and then unmount the boot node root file system. If *bootimage* is a raw device, skip this step. For example, if the *bootimage* file is `/bootimagedir/bootimage.new` and `bootroot_dir` is set to `/bootroot0`, type these commands.

```
smw:~ # mount /dev/bootrootdevice /bootroot0
smw:~ # cp -p /bootimagedir/bootimage.new /bootroot0/bootimagedir/bootimage.new
smw:~ # umount /bootroot0
```

3. Set the boot image for the next system boot using the suggested `xtcli` command.

   The `shell_bootimage_`*LABEL*`.sh` program suggests an `xtcli` command to set the boot image based on the value of `BOOT_IMAGE0` for the system set that you are using. The `-i` *bootimage* option specifies the path to the boot image and is either a raw device, for example, `/raw0` or `/raw1`, or a file such as `/bootimagedir/bootimage.new`.

   Caution: The next boot, anywhere on the system, uses the boot image you set here.

   a. Display the currently active boot image. Record the output of this command.

      If the partition variable in `CLEinstall.conf` is `s0`, type:

      ```
      smw:~# xtcli boot_cfg show
      ```

      Or

      If the partition variable in `CLEinstall.conf` is a partition value such as `p0`, `p1`, and so on, type:

      ```
      smw:~# xtcli part_cfg show pN
      ```

   b. Invoke `xtcli` with the `update` option to set the default boot configuration used by the boot manager.

      If the partition variable in `CLEinstall.conf` is `s0`, type this command to select the boot image to be used for the entire system.

      ```
      smw:~# xtcli boot_cfg update -i bootimage
      ```

      Or

If the partition variable in `CLEinstall.conf` is a partition value such as `p0`, `p1`, and so on, type this command to select the boot image to be used for the designated partition.

```
smw:~# xtcli part_cfg update pN -i bootimage
```

3.3.2 Customizing Existing Boot Images

Cray recommends using Procedure 2 on page 64 to prepare production boot images. However, you may use the `xtclone`, `xtpackage` and `xtbootimg` utilities on the SMW to modify existing compute node or service node images for the purpose of experimenting with custom options.

Note: You must have root privileges to invoke the `xtclone` and `xtpackage` commands.

You can customize a boot image on the SMW using a four-step process:

1. Execute the `xtclone` utility to create your new work area, which is copied from an existing work area.
2. In your new work area, make necessary changes, for example, install RPMs, edit configuration files, or add or remove scripts.
3. Execute the `xtpackage` utility to properly package the operating system components and prepare a load file for use by `xtbootimg`.
4. Execute the `xtbootimg` utility to create a boot image (an archive or `cpio` file) from your work area. The `xtbootimg` utility collects the components described by one or more load files into a single archive. The load files themselves are also included in the archive, along with other components, BIOS, and sources listed in the load file from `xtpackage`.

The following is a sample service node load file (`SNL0.load`):

```
#NODES_REALLY_READY
SNL0/size-initramfs 0x9021C
#Kernel source: /opt/xt-images/hostname-5.0.41-LABEL-s0/service/boot/\
bzImage-3.0.58-0.6.6.1_1.0500.7272-cray_ari_s
SNL0/bzImage-3.0.58-0.6.6.1_1.0500.7272-cray_ari_s.bin 0x100000
#Parameters source: /opt/xt-images/hostname-5.0.41-LABEL-s0/service/boot/parameters-snl
SNL0/parameters 0x90800
SNL0/initramfs.gz 0xFA00000
```

To create load files for supporting, for example, different boot parameters or different RAM disk contents, use the `xtpackage` command with the `-L` option.

Use the `xtbootimg -L` option to specify the path to the CNL compute node load file and the path to the service node load file.

Procedure 3. Creating a Cray boot image from existing file system images

1. Make copies of the compute-node-side and service-node-side of an existing work area.

   Note: It is recommended that your work area be in a subdirectory of `/opt/xt-images`, as shown in the example.

   ```
   smw:~ # xtclone /opt/xt-images/machine-xtrelease-LABEL/compute \
   /opt/xt-images/test/compute
   smw:~ # xtclone -s /opt/xt-images/machine-xtrelease-LABEL/service \
   /opt/xt-images/test/service
   ```

2. Make any changes to your work area that are necessary for your site. For example, you can install or erase RPMs, change configuration files, or add or remove scripts. Use the `xtpackage -s` option to create a "service-node-only" boot image. When you are finished making changes, wrap up (package) the compute-node-side and service-node-side of your work area.

   ```
   smw:~ # xtpackage /opt/xt-images/test/compute
   smw:~ # xtpackage -s /opt/xt-images/test/service
   ```

   Note: The `xtpackage` utility automatically creates an `/etc/xt.snl` file in service node `initramfs`. This allows compute node hardware to boot service node images, if necessary.

3. Finally, make a boot image (a `cpio` file) from your work area.

```
smw:~ # xtbootimg -L /opt/xt-images/test/service/SNL0.load \
-L /opt/xt-images/test/compute/CNL0.load \
-c /opt/xt-images/cpio/test/bootimage
```

Note: The directory path for *bootimage* **must** exist on both the SMW and the boot node, and the *bootimage* files in each location must be identical.

Some configurations export `/opt/xt-images/cpio` via NFS, so the SMW and the boot node can see the same files in `/opt/xt-images/cpio`, although this is not recommended for larger systems. Other configurations use a non-networked file system at `/tmp/boot`, in which case, you **must** put a copy of `smw:/tmp/boot/`*bootimage*`.cpio` at `boot:/tmp/boot/`*bootimage*. This is required for the boot node to be able to distribute *bootimage* to the other service nodes.

For more information about these utilities, see the `xtclone`(8), `xtpackage`(8), and `xtbootimg`(8) man pages.

3.3.3 Changing Boot Parameters

Caution: Some of the default boot parameters are mandatory. The system may not boot if they are removed.

Updating the parameters passed to the Linux kernel requires recreating the boot image with the `xtpackage` and `xtbootimg` commands. You can either edit the files in the file system image or specify a path to a file containing parameters. If editing the files, the default service and compute node parameters can be found in `boot/parameters-snl` and `boot/parameters-cnl`, respectively.

Example 2. Making a boot image with new parameters for service and CNL compute nodes

```
smw:~ # xtpackage -s -p /tmp/parameters-service.new  /opt/xt-images/test/service
smw:~ # xtpackage -p /tmp/parameters-compute.new /opt/xt-images/test/compute

smw:~ # xtbootimg -L /opt/xt-images/test/service/SNL0.load \
-L /opt/xt-images/test/compute/CNL0.load -c /raw0
```

3.4 Booting Nodes

This section describes how to manually boot your boot node, service nodes, and compute nodes. It also describes how to reboot a single compute node, and reboot login or network nodes.

For information about modifying boot automation files, see Modifying Boot Automation Files on page 189.

3.4.1 Booting the System

Use the `xtbootsys` command to manually boot your boot node, service nodes, and CNL compute nodes.

Note: You can also boot the system using both user-defined and built-in procedures in automation files, for example, `auto.generic.cnl`. Before you modify the `auto.generic.cnl` file, Cray recommends copying it first because it will be replaced by an SMW software upgrade. For related procedures, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

The `xtbootsys` command prevents unintentional booting of currently booted partitions. If a boot automation file is being used, `xtbootsys` checks that file to determine if the string `shutdown` exists within any actions defined in the file. If it does, then `xtbootsys` assumes that a shutdown is being done, and no further verification of operating on a booted partition occurs. If the partition is not being shut down, and the boot node is in the `ready` state, then `xtbootsys` announces this fact to you and queries you for confirmation that you want to proceed. By default, confirmation is enabled. To disable or enable confirmation when booting booted partitions, use the `xtbootsys config,confirm_booting_booted` and the `config,confirm_booting_booted_last_session` global TCL variables, the `--config` *name=value* on the `xtbootsys` command line, or the `XTBOOTSYS_CONFIRM_BOOTING_BOOTED` and `XTBOOTSYS_CONFIRM_BOOTING_BOOTED_LAST_SESSION` environment variables.

Procedure 4. Manually booting the boot node and service nodes

Note: The Lustre file system should start up before the compute nodes, and compute node Lustre clients should be unmounted before shutting down the Lustre file system.

Note: If you run more than one boot image, you can check which image you are set up to boot with the `xtcli boot_cfg show` or `xtcli part_cfg show p`*N* commands. To change which image you are booting, see Updating Boot Configuration on page 188.

1. As `crayadm`, use the `xtbootsys` command to boot the boot node.

   ```
   crayadm@smw:~> xtbootsys
   ```

   Note: If you have a partitioned system, invoke `xtbootsys` with the `--partition p`*n* option.

   The `xtbootsys` command prompts you with a series of questions. Cray recommends that you answer `yes` by typing **Y** to each question.

The session pauses at:

```
Enter your boot choice:
       0) boot bootnode ...
       1) boot sdb ...
       2) boot compute ...
       3) boot service ...
       4) boot all (not supported) ...
       5) boot all_comp ...
       10) boot bootnode and wait ...
       11) boot sdb and wait ...
       12) boot compute and wait ...
       13) boot service and wait ...
       14) boot all and wait (not supported) ...
       15) boot all_comp and wait ...
       17) boot using a loadfile ...
       18) turn console flood control off ...
       19) turn console flood control on ...
       20) spawn off the network link recovery daemon (xtnlrd)...
       q) quit.
```

Choose option **10** to boot the boot node and wait.

You are prompted to confirm your selection. Press the `Enter` key or type **Y** to each question to confirm your selection.

```
Do you want to boot the boot node ? [Yn] Y
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
```

2. After the boot node has finished booting, the process returns to the boot choice menu. Choose option **11** to boot the SDB node and wait.

   You are prompted to confirm your selection. Press the `Enter` key or type **Y** to each question to confirm your selection.

```
Do you want to boot the sdb node ? [Yn] Y
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
```

3. Next, select option **13** to boot the service nodes and wait.

   You are prompted to enter a list of the service nodes to be booted. Type `p0` to boot the remaining service nodes in the entire system, or `p`*N* (where *N* is the partition number) to boot a partition.

4. To confirm your selection, press the `Enter` key or type **Y** to each question.

```
Do you want to boot service p0 ? [Yn] Y
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
```

5. Log on to any service nodes for which there are local configuration or startup scripts (such as starting Lustre) and run the scripts.

Procedure 5. Booting the compute nodes

1. After all service and login nodes are booted and Lustre has started (if configured at this time), return to the `xtbootsys` menu.

2. Select **17** from the `xtbootsys` menu. A series of prompts are displayed. Type the responses indicated in the following example. For the `component list` prompt, type **p0** to boot the entire system, or **p***N* (where *N* is the partition number) to boot a partition. At the final three prompts, press the `Enter` key.

```
Enter your boot choice: 17
Enter a boot type string (or nothing to do nothing): CNL0
Enter a boot type option (or nothing to do nothing): compute
Enter a component list (or nothing to do nothing): p0
Enter 'any' to wait for any console output,
   or 'linux' to wait for a linux style boot,
   or anything else (or nothing) to not wait at all: Enter
Enter an alternative CPIO archive name (or nothing): Enter
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Enter
```

3. After all the compute nodes are booted, return to the `xtbootsys` menu. Type **q** to exit the `xtbootsys` program.

4. Remove the `/etc/nologin` file from all service nodes to permit a non-root account to log on.

```
smw:~# ssh root@boot
boot:~# xtunspec -r /rr/current -d /etc/nologin
```

3.4.2 Using the `xtcli boot` Command to Boot a Node or Set of Nodes

To boot a specific image or load file on a given node or set of nodes, you can execute the HSS `xtcli boot` *boot_type* command, as shown in the following examples.

Warning: Each system boot must be started with an `xtbootsys` session to establish a *sessionid*. Only perform direct boot commands using the `xtcli boot` command after a session has been established through `xtbootsys`.

Note: When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.

Example 3. Booting all service nodes with a specific image

The following example boots all service nodes with the specific image located at `/raw0`:

```
crayadm@smw:~> xtcli boot all_serv_img -i /raw0
```

Example 4. Booting all compute nodes with a specific image

The following example boots all compute nodes with the specific image located at */bootimagedir/bootimage*:

```
crayadm@smw:~> xtcli boot all_comp_img -i /bootimagedir/bootimage
```

Example 5. Booting compute nodes using a load file

The following example boots all compute nodes in the system with using a load file name `CNL0`:

```
crayadm@smw:~> xtcli boot CNL0 -o compute s0
```

3.4.3 Rebooting a Single Compute Node

You can initiate a warm boot with the `xtbootsys` command's `--reboot` option. This operation performs minimal initialization followed by a boot of only the selected compute nodes. Unlike the sequence that is used by the `xtbounce` command, there is no power cycling of the Cray ASICs or of the node itself, so the high-speed network (HSN) routing information is preserved. Do not specify a session identifier (`-s` or `--session` option) because `--reboot` continues the last session and adds the selected components to it.

Example 6. Rebooting a single compute node

```
crayadm@smw:~> xtbootsys --reboot c1-0c2s1n2
```

3.4.4 Rebooting Login or Network Nodes

Login or network nodes cannot be rebooted through a `shutdown` or reboot command issued on the node; they must be restarted through the HSS system using the `xtbootsys --reboot` *idlist* SMW command. The HSS must be used so that the proper kernel is pushed to the node.

Note: Do not attempt to warm boot nodes running other services in this manner.

Example 7. Rebooting login or network nodes

```
crayadm@smw:~> xtbootsys --reboot idlist
```

For additional information, see the `xtbootsys`(8) man page.

3.5 Rebooting Controllers of a Cabinet or Blade

The `xtccreboot` command provides a means to reboot controllers. Options allow for rebooting all controllers of a specified type (cabinet or blade) or providing a list of controllers of a specified type to be rebooted.

Example 8. Rebooting cabinet controller `c0-0`, with verbose output

```
smw:~> xtccreboot -v -c c0-0
xtccreboot: /opt/cray-xt-pdsh/default/bin/pdsh -w "c0-0" /sbin/reboot
xtccreboot: reboot sent to specified CCs
```

For additional information, see the `xtccreboot`(8) man page.

3.6 Requesting and Displaying System Routing

Use the HSS `rtr` command to request routing for the HSN, to verify your current route configuration, or to display route information between nodes. Upon startup, `rtr` determines whether it is making a routing request or an information request.

Example 9. Displaying routing information

The `--system-map` option to `rtr` writes the current routing information to `stdout` or to a specified file. This command can also be helpful for translating node IDs (NIDs) to physical ID names.

```
crayadm@smw:~> rtr --system-map
```

Example 10. Routing the entire system

The `rtr -R|--route-system` command sends a request to perform system routing. If no components are specified, the entire configuration is routed as a single routing domain based on the configuration information provided by the state manager. If a component list (*idlist*) is provided, routing is limited to the listed components. The state manager configuration further limits the routing domain to omit disabled blades, nodes, and links and empty blade slots.

```
crayadm@smw:~> rtr --route-system
```

For more information about displaying system routing information, see the `rtr`(8) man page.

3.7 Initiating a Network Discovery Process

Use the HSS `rtr --discover` command to initiate a network discovery process.

```
crayadm@smw:~> rtr --discover
```

> **Important:** The discovery process must be done on the system as a whole; it cannot be applied to individual partitions. Therefore, discovery will immediately fail if the system does not have partition `p0` enabled.

See the `rtr`(8) man page for additional information about using the `rtr --discover` command.

3.8 Bouncing Blades Repeatedly Until All Blades Succeed

Example 11. Bounce failed blades repeatedly until all blades succeed

To bounce failed blades multiple times in order to have them eventually all succeed, complete these steps.

Important: Doing this iterative `xtbounce` should typically be done in concert with an `xtbootsys` automation file where bounce and routing are turned off.

1. Bounce the system.

   ```
   smw:~> xtbounce s0
   ```

2. Bounce any blades that failed the first bounce.

3. Repeat step 2 as necessary.

4. Execute the following command, which copies route configuration files, based on the *idlist* (such as `s0`), to the blade controllers. This avoids having old, partial route configuration files left on the blades that were bounced in step 2 above and ensures that the links are initialized correctly. For example,

   ```
   smw:~> xtbounce --linkinit s0
   ```

5. Route and boot the system, without executing `xtbounce` again; if using a `xtbootsys` automation file, specify `set data(config,xtbounce) 0`, or you can use the `xtbootsys --config xtbounce=0` command.

3.9 Shutting Down the System Using the `auto.xtshutdown` File

The preferred method to shut down the system is to use the `xtbootsys -s last -a auto.xtshutdown` command. This method shuts down the compute nodes (which are commonly also Lustre clients), then executes `xtshutdown` on service nodes, halting the nodes and then stopping processes on the SMW. You can shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file, which is located on the SMW in the `/opt/cray/hss/default/etc` directory.

Example 12. Shutting down the system using the `auto.xtshutdown` file

To shut down the system using the `auto.xtshutdown` file, execute the following command from the SMW:

```
crayadm@smw:~> xtbootsys -s last -a auto.xtshutdown
```

Or

for a partitioned system with partition p*N*:

```
smw:~# xtbootsys --partition pN -s last -a auto.xtshutdown
```

For related procedures, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444). For more information about using automation files, see the `xtbootsys`(8) man page.

3.10 Shutting Down Service Nodes Using the `xtshutdown` Command

The `xtshutdown` command executes from the boot node to shut down the services on service nodes and then shut down the service nodes of the Cray system. It executes a series of commands locally on the boot node and on the service nodes to shut down the system in an orderly fashion.

Procedure 6. Shutting down service nodes

- Modify the `/etc/opt/cray/init-service/xtshutdown.conf` file or in the file specified by the `XTSHUTDOWN_CONF` environment variable to define the sequence of shutdown steps and the nodes on which to execute them. (The `/etc/opt/cray/init-service/xtshutdown.conf` file resides on the boot node.)

Caution: The `xtshutdown` command does not shut down compute nodes. To shut down the compute and service nodes, see Shutting Down the System or Part of the System Using the `xtcli shutdown` Command on page 77.

The `xtshutdown` command uses `pdsh` to invoke commands on the service nodes you select. You can choose the boot node, SDB node, a class of nodes, or a single host. You can define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

Note: You must be `root` user to use the `xtshutdown` command. Passwordless `ssh` must be enabled for the root user from the boot node to all service nodes.

```
boot:~ # xtshutdown
```

After you have shut down the software on the nodes, you can halt the hardware, reboot, or power down.

For information about shutting down service nodes, see the `xtshutdown`(8) man page.

3.11 Shutting Down the System or Part of the System Using the `xtcli shutdown` Command

The HSS `xtcli shutdown` command allows you to shut down the system or a part of the system. To shut down compute nodes, execute the `xtcli shutdown` command. Under normal circumstances, for example to successfully disconnect from Lustre, invoking the `xtcli shutdown` command attempts to gracefully shut down the specified nodes.

Example 13. Shutting down all compute nodes

To gracefully shut down all compute nodes, execute the following command:

```
crayadm@smw:~> xtcli shutdown compute
```

Example 14. Shutting down specified compute nodes

To gracefully shut down only compute nodes in cabinet `c13-2`:

```
crayadm@smw:~> xtcli shutdown c13-2
```

Example 15. Shutting down all nodes of a system

The `xtcli shutdown` command allows you to shut down the system gracefully; to shut down a partition, use the `p`*n* command, where *n* is the partition you want to shut down.

```
crayadm@smw:~> xtcli shutdown s0
```

Example 16. Forcing nodes to shut down (immediate halt)

To force nodes to shut down, for example when all nodes of a system must be halted immediately, use the `-f` argument; nodes will not go through their normal shutdown process. You can force a shutdown by using the `-f` argument, even if the nodes have an alert status present. For example:

```
crayadm@smw:~> xtcli shutdown -f s0
```

After you have shut down the software on the nodes, you can halt the hardware, reboot, or power down.

For information about shutting down nodes using the `xtcli shutdown` command, see the `xtcli`(8) man page.

3.12 Stopping System Components

When you remove, stop, or power down components, any applications and compute processes that are running on those components are lost.

3.12.1 Reserving a Component

If you want the applications and compute processes to complete before you stop components, use the HSS `xtcli set_reserve` *idlist* command to select the nodes you want to remove. This prevents them from accepting new jobs.

Note: If you are running CNL and using ALPS, after a node is reserved it is considered to be `down` by ALPS. The output from `apstat` will show the node as down (DN), even though there may be an application running on that node. This DN designation indicates that no other work will be placed on the node after the currently running application has terminated.

Procedure 7. Reserving a component

- Type:

```
crayadm@smw:~> xtcli set_reserve idlist
```

For information about reserving a component, see the `xtcli_set`(8) man page.

3.12.2 Powering Down Blades or Cabinets

Warning: Power down the cabinets with software commands. Tripping the circuit breakers may result in damage to system components.

Warning: Ensure the operating system is not running before you power down a blade or a cabinet.

The `xtcli power down` command powers down the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the READY state (see Appendix B, System States on page 393) to receive power commands.

The `xtcli power force_down` and `xtcli power down_slot` commands are aliases for the `xtcli power down` command.

Procedure 8. Powering down a specified blade

The `xtcli power down` command has the form, where *physIDlist* is a comma-separated list of cabinets or blades present on the system (see Physical ID on page 53).

```
xtcli power down physIDlist
```

- To power down a blade with the ID `c0-0c0s7`, type:

```
crayadm@smw:~> xtcli power down c0-0c0s7
```

Warning: Although a blade is powered off, the HSS in the cabinet is live and has power.

For information about disabling and enabling components, see Disabling Hardware Components on page 84, and Enabling Hardware Components on page 84, respectively. For information about powering down a component, see the `xtcli_power`(8) man page.

3.12.3 Halting Selected Nodes

You can halt selected nodes with the HSS `xtcli halt` command.

Procedure 9. Halting a node

The command has the form:

`xtcli halt` *node*

- Type:

  ```
  crayadm@smw:~> xtcli halt node
  ```

For more information about halting a node, see the `xtcli`(8) man page.

3.13 Restarting a Blade or Cabinet

Note: Change the state of the hardware only when the operating system is not running or is shut down.

The `xtcli power up` command powers up the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the READY state (see Appendix B, System States on page 393) to receive power commands.

The `xtcli power up` command does not attempt to power up network mezzanine cards or nodes, which are handled by the `xtbounce` command during system boot. The `xtcli power up_slot` command is an alias for the `xtcli power up` command.

Use the HSS `xtcli power up` command to restart a component.

Procedure 10. Power up blades in a cabinet

The `xtcli power up` command has the form, where *physIDlist* is a comma-separated list of cabinets or blades present on the system (see Physical ID on page 53).

`xtcli power up` *physIDlist*

- Power up the selected blade:

  ```
  crayadm@smw:~> xtcli power up blade
  ```

For more information, see the `xtcli_power`(8) man page.

3.14 Aborting Active Sessions on the HSS Boot Manager

Use the HSS `xtcli session abort` command to abort sessions in the boot manager. A session corresponds to executing a specific command such as `xtcli power up` or `xtcli boot`.

Example 17. Aborting a session running on the boot manager

To display all running sessions in the boot manager, execute the following command.

```
crayadm@smw:~> session show BM all
```

Execute the following HSS `xtcli session abort` command to abort session 1 running on the boot manager:

```
crayadm@smw:~> xtcli session abort BM 1
```

Use this command if you have started an `xtcli power` or `xtcli boot` command but want to stop it before the command has completed.

Note: Only the boot manager supports multiple simultaneous sessions.

For more information about manager sessions, see the `xtcli`(8) man page.

3.15 Displaying and Changing Software System Status

There are a number of tools that enable you to inspect and change the status of compute nodes on a running system.

3.15.1 Displaying the Status of Nodes from the Operating System

The user command `xtnodestat` provides a display of the status of nodes: how they are allocated and to what jobs. The `xtnodestat` command provides current job and node status summary information, and it provides an interface to ALPS and jobs running on CNL compute nodes. You must be running ALPS in order for `xtnodestat` to report job information.

For more information, see the `xtnodestat`(1) man page.

3.15.2 Viewing and Changing the Status of Nodes

Use the `xtprocadmin` command on a service node to view the status of components of a booted system in the `processor` table of the SDB. The command enables you to retrieve or set the processing mode (`interactive` or `batch`) of specified nodes. You can display the state (`up`, `down`, `admindown`, `route`, or `unavailable`) of the selected components, if needed. You can also allocate processor slots or set nodes to become unavailable at a particular time. The node is scheduled only if the status is `up`.

Example 18. Looking at node characteristics

```
login:~> xtprocadmin
   NID    (HEX)    NODENAME     TYPE    STATUS        MODE
     1      0x1  c0-0c0s0n1  service        up       batch
     2      0x2  c0-0c0s0n2  service        up       batch
     5      0x5  c0-0c0s1n1  service        up       batch
     6      0x6  c0-0c0s1n2  service        up       batch
     8      0x8  c0-0c0s2n0  compute        up       batch
     9      0x9  c0-0c0s2n1  compute        up       batch
    10      0xa  c0-0c0s2n2  compute        up       batch
    11      0xb  c0-0c0s2n3  compute        up       batch
```

Example 19. Viewing all node attributes

Use the `xtprocadmin` command to view current node attributes. The `xtprocadmin -A` option lists all attributes of selected nodes. For example:

```
login:~> xtprocadmin -A
   NID    (HEX)    NODENAME     TYPE ARCH        OS CPUS  CU AVAILMEM   PAGESZ CLOCKMHZ GPU SOCKETS DIES C/CU
     1      0x1  c0-0c0s0n1  service   xt (service)   16   8    32768     4096     2600   0       1    1    2
     2      0x2  c0-0c0s0n2  service   xt (service)   16   8    32768     4096     2600   0       1    1    2
     5      0x5  c0-0c0s1n1  service   xt (service)   16   8    32768     4096     2600   0       1    1    2
     6      0x6  c0-0c0s1n2  service   xt (service)   16   8    32768     4096     2600   0       1    1    2
     8      0x8  c0-0c0s2n0  compute   xt       CNL   32  16    65536     4096     2600   0       2    2    2
     9      0x9  c0-0c0s2n1  compute   xt       CNL   32  16    65536     4096     2600   0       2    2    2
    10      0xa  c0-0c0s2n2  compute   xt       CNL   32  16    65536     4096     2600   0       2    2    2
```

Example 20. Viewing selected node attributes of selected nodes

The `xtprocadmin -a` *attr1,attr2* option lists selected attributes of selected nodes. For example:

```
login:~> xtprocadmin -n 8 -a arch,clockmhz,os,cores
   NID    (HEX)    NODENAME     TYPE ARCH CLOCKMHZ        OS CPUS
     8      0x8  c0-0c0s2n0  compute   xt     2600       CNL   32
```

Example 21. Disabling a node

To mark a node as `admindown` and not allow it to be allocated, type the following command:

```
crayadm@nid00004:~> xtprocadmin -n c0-0c0s3n1 -k s admindown
```

Example 22. Disabling all processors

To mark all processors as `admindown` and to disable the system's ability to change their state, type the following command:

```
crayadm@nid00004:~> xtprocadmin -k s admindown
```

Note: When the `xtprocadmin -ks` option is used, then the option can either a normal argument (`up`, `down`, etc.), or it can have a colon in it to represent a conditional option; for example, the option of the form `up:down` means "if state was up, mark down".

For more information, see the `xtprocadmin`(8) man page.

3.15.3 Marking a Compute Node as a Service Node

Use the `xtcli mark_node` command to mark a node in a compute blade to have a role of `service` or `compute`; `compute` is the default. It is **not** permitted to change the role of a node on a service blade, which always has the `service` role.

Marking a node on a compute blade as `service` or `compute` allows you to load the desired boot image at boot time. Compute nodes marked as `service` can run software-based services. A request to change the role of a running node (that is, the node is in the `ready` state and the operating system is running) will be denied.

For more information, see the `xtcli`(8) man page and Checking the Status of System Components on page 100.

3.15.4 Finding Node Information

3.15.4.1 Translating Between Physical ID Names and Integer NIDs

To translate between physical ID names (cnames) and integer NIDs, generate a system map by using the `rtr --system-map` command on the SMW and filter the output; for example:

```
crayadm@smw:~> rtr --system-map | grep cname | awk '{ print $1 }'
```

For more information, see the `rtr`(8) man page.

3.15.4.2 Finding Node Information Using the `xtnid2str` Command

The `xtnid2str` command converts numeric node identification values to their physical names (cnames). This allows conversion of Node ID values, ASIC NIC address values, or ASIC ID values.

Example 23. Finding the physical ID for node 38

```
smw:~> xtnid2str 28
node id 0x26 = 'c0-0c0s1n2'
```

Example 24. Finding the physical ID for nodes 0, 1, 2, and 3

```
smw:~> xtnid2str 0 1 2 3
node id 0x0 = 'c0-0c0s0n0'
node id 0x1 = 'c0-0c0s0n1'
node id 0x2 = 'c0-0c0s1n0'
node id 0x3 = 'c0-0c0s1n1'
```

Example 25. Finding the physical IDs for Aries IDs 0-7

```
smw:~> xtnid2str -a 0-7
aries id 0x0 = 'c0-0c0s0a0'
aries id 0x1 = 'c0-0c0s1a0'
aries id 0x2 = 'c0-0c0s2a0'
aries id 0x3 = 'c0-0c0s3a0'
aries id 0x4 = 'c0-0c0s4a0'
aries id 0x5 = 'c0-0c0s5a0'
aries id 0x6 = 'c0-0c0s6a0'
aries id 0x7 = 'c0-0c0s7a0'
```

For additional information, see the `xtnid2str`(8) man page.

3.15.4.3 Finding Node Information Using the `nid2nic` Command

Use the `nid2nic` command to print the *nid*-to-*nic* address mappings, *nic*-to-*nid* address mappings, and a specific *physical_location*-to-*nic* address and *nid* mappings.

For information about using the `nid2nic` command, see the `nid2nic`(8) man page.

Example 26. Printing the *nid*-to-*nic* address mappings for the node with NID 31.

```
smw:~> nid2nic 31
NID:0x1f        NIC:0x1f        c0-0c0s7n3
```

Example 27. Printing the *nid*-to-*nic* address mappings for the same node as shown in Example 26, but specifying the NIC value in the command line

```
smw:~> nid2nic -n 0x21
NIC:0x21        NID:0x21        c0-0c0s8n1
```

3.16 Displaying and Changing Hardware System Status

You can execute commands that look at and change the status of the hardware.

Caution: Execute commands that change the status of hardware only when the operating system is shut down.

3.16.1 Generating HSS Physical IDs

Execute the HSS `xtgenid` command to generate HSS physical IDs, for example, to create a list of blade controller identifiers for input to the flash manager. You can restrict your selections to components that are of a particular type.

Note: Only user `root` can execute the `xtgenid` command.

Example 28. Creating a list of node identifiers that are not in the `DISABLE`, `EMPTY`, or `OFF` state

```
smw:~ # xtgenid -t node --strict
```

For more information, see the `xtgenid`(8) man page.

3.16.2 Disabling Hardware Components

If links, nodes, or Cray ASICs have hardware problems, you can direct the system to ignore the components with the `xtcli disable` command.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

Important: The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Disabling of a node in the `ready` state will fail, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

The state of `empty` components will not change when using the `disable` command, unless the force option (`-f`) is used.

The `xtcli disable` command has the form:

`xtcli disable [{-t` *type* `[-a] } | -n][-f]` *idlist*

where *idlist* is a comma-separated list of components (in cname format) that you want the system to ignore. The system disregards these links or nodes.

Example 29. Disabling the Aries ASIC `c0-0c1s3a0`

1. Determine that the ASIC is in the `OFF` state.

   ```
   crayadm@smw:~> xtcli status -t aries c0-0c1s3a0
   ```

2. If the ASIC is not in the `OFF` state, power down the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power down c0-0c1s3
   ```

3. Disable the ASIC.

   ```
   crayadm@smw:~> xtcli disable c0-0c1s3a0
   ```

4. Power up the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power up c0-0c1s3
   ```

For detailed information about using the `xtcli disable` command, see the `xtcli`(8) man page.

3.16.3 Enabling Hardware Components

If links, nodes, or Cray ASICs that have been disabled are later fixed, you can add them back to the system with the `xtcli enable` command.

By default, when enabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

Important: The `-n` option with the `xtcli enable` command must be used carefully because this may create invalid system state configurations.

The state of `empty` components does not change when using the `xtcli enable` command, unless the force option (`-f`) is used.

The `xtcli enable` command has the form:

```
xtcli enable [{-t type [-a] } | -n][-f] idlist
```

where *idlist* is a comma-separated list of components (in cname format) that you want the system to recognize.

The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If you disable a component, the state shown becomes `disabled`. When you use the `xtcli enable` command to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

For more information, see the `xtcli`(8) man page.

3.16.4 Setting Components to Empty

Use the `xtcli set_empty` command to set a selected component to the `empty` state. HSS managers and the `xtcli` command ignore empty or disabled components.

Setting a selected component to the `empty` state is typically done when a component, usually a blade, is physically removed. By setting it to `empty`, the system ignores it and routes around it.

By default, when setting a component to an `empty` state, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

Note: The `-n` option with the `set_empty` command must be used carefully because this may create invalid system state configurations.

Example 30. Setting a blade to the empty state

Set the blade and all its components to `empty`:

```
crayadm@smw:~> xtcli set_empty -a c0-0c1s7
```

For more information, see the `xtcli`(8) man page.

3.16.5 Locking Components

Components are automatically locked when a command that can change their state is running. As the command is started, the state manager locks these components so that nothing else can affect their state while the command executes. When the manager is finished with the command, it unlocks the components.

Use the HSS `xtcli lock` command to lock components.

Example 31. Locking cabinet `c0-0`

The lock command identifies the session ID. Locking a component prints out the state manager session ID.

```
crayadm@smw:~> xtcli lock -l c0-0
```

Example 32. Show all session (lock) data

You can use the `xtcli lock show` command to show session (lock) information.

```
crayadm@smw:~> xtcli lock show
```

3.16.6 Unlocking Components

Use the HSS `xtcli lock` command to unlock components.

Example 33. Unlocking cabinet `c0-0`

The `xtcli lock` command is useful when a manager fails to unlock some set of components. You can manually check for locks with the `xtcli lock show` command and unlock them. Unlocking a component does not print out the state manager session ID. The `-u` option must be used to unlock a component.

```
crayadm@smw:~> xtcli lock -u lock_number
```

lock_number is the value given when initiating the lock; it is also indicated in the `xtcli lock show` query. Unlocking does nothing to the state of the component other than to release locks associated with it. HSS managers cannot affect components that are locked by a different session.

3.17 Performing Parallel Operations on Service Nodes

Use the `pdsh` command, which is the CLE parallel remote shell utility, on a service node to issue commands to groups of nodes in parallel. You can select the nodes on which to use the command, exclude nodes from the command, and limit the time the command is allowed to execute. You must be user `root` to execute the `pdsh` command. The command has the form:

`pdsh [`*options*`]` *command*

Example 34. Restarting the NTP service

To restart the network time protocol (NTP) service on the first 9 login nodes, type:

```
boot:~ # pdsh -w 'login[1-9]' /etc/init.d/ntp restart
```

For more information, see the pdsh(1) man page.

3.18 Performing Parallel Operations on Compute Nodes

The parallel command tool (pcmd) allows you to execute the same commands on compute nodes in parallel, similar to pdsh. (You launch the pcmd command from a service node, but it acts on compute nodes.) It allows administrators and/or, if your site deems it feasible, other users to securely execute programs in parallel on compute nodes. You can specify on which nodes to execute the command. Alternatively, you can specify an application ID (*apid*) to execute the command on all the nodes available under that *apid*.

An unprivileged user must execute the command targeting nodes where the user is currently running an aprun. A root user is allowed to target any compute node, regardless of whether there are jobs running there or not. In either case, if the aprun exits and the associated applications are killed, any commands launched by pcmd will also exit.

By default, pcmd is installed as a root-only tool. It must be installed as setuid root in order for unprivileged users to use it.

The pcmd command is located in the nodehealth module. If the nodehealth module is not part of your default profile, you can load it by specifying:

```
module load nodehealth
```

For additional information, see the pcmd(1) man page.

3.19 xtbounce Error Message Indicating Cabinet Controller and Its Blade Controllers Not in Sync

During the gather_cab_pwr_states phase of xtbounce, if the HSS software on a cabinet controller and any of its blade controllers is out of sync, error messages such as the following will be printed during the xtbounce:

```
***** gather_cab_pwr_states *****
18:28:42 - Beginning to wait for response(s)

ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
```

If this occurs, it indicates that the blade controller software is at a different revision than the cabinet controller software. xtbounce will print a list of cabinets for which this error has occurred. The message will be like:

```
ERROR: power state check error on 2 cabinet(s)
WARNING: unable to find c0-0 in err_cablist
WARNING: unable to find c0-2 in err_cablist
```

This error is an indication that when the HSS software was previously updated, the cabinet controllers and the blade controllers were not updated to the same version.

To correct this error, cancel out of xtbounce (with Ctrl-C), wait approximately five minutes for the xtbounce related activities on the blade controllers to finish, then reboot the cabinet controller(s) and their associated blade controllers to get the HSS software synchronized. Following this, the xtbounce may be executed once again.

3.20 Handling Bus Errors

Bus errors are caused by machine-check exceptions. If you have received a bus error, try the following procedure.

Procedure 11. Power-cycling a component

Power down then power up components. The *physIDlist* is a comma-separated list of components present on the system (see Physical ID on page 53).

1. Power down the components.

   ```
   crayadm@smw:~> xtcli power down physIDlist
   ```

2. Power up the components.

   ```
   crayadm@smw:~> xtcli power up physIDlist
   ```

3.21 Handling Component Failures

Components that fail are replaced as field replaceable units (FRUs). FRUs include compute blade components, service blade components, and power and cooling components.

When a field replaceable unit (FRU) problem arises, contact your Customer Service Representative to schedule a repair.

3.22 Capturing and Analyzing System-level and Node-level Dumps

3.22.1 Dumping Information Using the `xtdumpsys` Command

The `xtdumpsys` command collects and analyzes information from a Cray system that is failing or has failed, has crashed, or is hung. Analysis is performed on, for example, event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors. When failed components are found, detailed information is gathered from them.

To collect similar information for components that have not failed, invoke the `xtdumpsys` command with the `--add` option and name the components from which to collect data. The HSS `xtdumpsys` command saves dump information in `/var/opt/cray/dump/`*timestamp* by default.

Example 35. Dumping information about a working component

To dump the entire system and collect detailed information from all blade controllers in chassis `0` of cabinet `0`, type:

```
crayadm@smw:~> xtdumpsys --add c0-0c0s0
```

Note: When using the `--add` option to add multiple components, separate components with spaces, not commas.

Effective with the 7.0.UP03 release of the Cray SMW software, the `xtdumpsys` has been rewritten. The previous implementation is still supported and has been renamed `xtdumpsys-old`.

The new `xtdumpsys` command is written in Python and support plug-ins written in Python. A number of plug-in scripts are included in the software release. Call `xtdumpsys --list` to view a list of included plug-ins and their respective directories.

The `xtdumpsys` command also now supports the use of configuration files to specify `xtdumpsys` presets, rather than entering them via the command line.

For more information, see the `xtdumpsys`(8) man page.

3.22.2 cdump and crash Utilities for Node Memory Dump and Analysis

The cdump and crash utilities may be used to analyze the memory on any Cray service node or CNL compute node. The cdump command is used to dump node memory to a file. After cdump completes, you may then use the crash utility on the dump file generated by cdump.

Cray recommends executing the cdump utility only if a node has panicked or is hung, or if a dump is requested by Cray.

To select the desired access method for reading node memory, use the cdump -r *access* option. Valid access methods are:

- xt-bhs: The xt-bhs method uses a basic hardware system server that runs on the SMW to access and read node memory. xt-bhs is the default access method for these systems.
- xt-hsn: The xt-hsn method utilizes a proxy that reads node memory through the High-speed Network (HSN). The xt-hsn method is faster than the xt-bhs method, but there are situations where it will not work (for example, if the ASIC is not functional). However, the xt-hsn method is preferable because the dump completes in a short amount of time and the node can be returned to service sooner.
- xt-file: The xt-file method is used for memory dump file created by the -z option. The compressed memory dump file must be uncompressed prior to executing this command. Use the file name for *node-id*.
- xc-knc: The xc-knc method is used to dump Intel Xeon Phi nodes. Use this method when dumping only the Xeon Phi coprocessor without dumping the host node. When dumping the host node, do not use xc-knc. A host node dump automatically includes dumping the Xeon Phi coprocessors unless they are suppressed by specifying the -n option.

To dump Cray node memory, *access* takes the following form:

method[@*host*]

For additional information, see the cdump(8) and crash(8) man pages.

3.22.3 Using dumpd to Automatically Dump and Reboot Nodes

The SMW daemon dumpd initiates automatic dump and reboot of nodes when requested by Node Health Checker (NHC).

Caution: The dumpd daemon is invoked automatically by xtbootsys on system (or partition) boot. In most cases, system administrators do not need to use this daemon directly.

You can set global variables in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file to control the interaction of NHC and `dumpd`. For more information about NHC and the `nodehealth.conf` configuration file, see Configuring Node Health Checker (NHC) on page 160.

You can also set variables in the `/etc/opt/cray-xt-dumpd/dumpd.conf` configuration file on the SMW to control how `dumpd` behaves on your system.

Each CLE release package also includes an example `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf.example`. The `dumpd.conf.example` file is a copy of the `/etc/opt/cray-xt-dumpd/dumpd.conf` file provided for an initial installation.

Important: The `/etc/opt/cray-xt-dumpd/dumpd.conf` file is not overwritten during a CLE upgrade if the file already exists. This preserves your site-specific modifications previously made to the file. However, you should compare your `/etc/opt/cray-xt-dumpd/dumpd.conf` file content with the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file provided with each release to identify any changes, and then update your `/etc/opt/cray-xt-dumpd/dumpd.conf` file accordingly.

If the `/etc/opt/cray-xt-dumpd/dumpd.conf` file does **not** exist, then the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file is copied to the `/etc/opt/cray-xt-dumpd/dumpd.conf` file.

The CLE installation and upgrade processes automatically install `dumpd` software but you must explicitly enable it.

3.22.3.1 Enabling dumpd

Procedure 12. Enabling dumpd

1. In the `nodehealth.conf` configuration file on the shared root (located in `/etc/opt/cray/nodehealth/nodehealth.conf`) change:

   ```
   dumpdon: off
   ```

   to

   ```
   dumpdon: on
   ```

   This allows node health to make requests to `dumpd`.

2. In the same file, set the `maxdumps` variable to some number greater than zero if you want dumps to be taken.

3. Specify an action of `dump`, `reboot`, or `dumpreboot` for any tests for which you want NHC to make a request of `dumpd` when that test fails.

4. In `dumpd.conf` configuration file on the SMW (in `/etc/opt/cray-xt-dumpd/dumpd.conf`), change:

   ```
   enable: no
   ```

   to

   ```
   enable: yes
   ```

After the changes to the configuration files are made, NHC will request action from `dumpd` for any test that fails with an action of `dump`, `reboot`, or `dumpreboot`.

3.22.3.2 `/etc/opt/cray-xt-dumpd/dumpd.conf` Configuration File

The `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf`, is located on the SMW. There is no need for you to change any installation configuration parameters; however, you may edit the `/etc/opt/cray-xt-dumpd/dumpd.conf` file to customize how `dumpd` behaves on your system using the following configuration variables.

`enable: yes|no`

Provides a quick on/off switch for all `dumpd` functionality (the default value in the file provided from Cray Inc. is `no`.)

`partitions:` *number*

Specifies whether or not `dumpd` acts on specific partitions or ranges of partitions. Placing `!` in front of a partition or range disables it.

For example, specifying

```
partitions: 1-10,!2-4
```

enables partitions 1, 5, 6, 7, 8, 9, and 10 but not 2, 3, or 4. Partitions **must** be explicitly enabled. Leaving this option blank disables all partitions.

`disabled_action: ignore|queue`

Specifies what to do when requests come in for a disabled partition. If you specify `ignore`, requests are removed from the database

and not acted upon. If you specify `queue`, requests continue to build while `dumpd` is disabled on a partition. When the partition is reenabled, the requests will be acted on. Specifying `queue` is not recommended if `dumpd` will be disabled for long periods of time, as it can cause SMW stress and database problems.

`save_output: always|errors|never`

Indicates when to save `stdout` and `stderr` from `dumpd` commands that are executed. If `save_output` is set to `always`, all output is saved. If `errors` is specified, output is saved only when the command exits with a nonzero exit code. If `never` is specified, output is never saved.

The default is to save output on errors.

`command_output:` *directory*

Specifies where to save output of `dumpd` commands, per the `save_output` variable. The command output is put in the file *action.pid.timestamp*`.out` in the directory specified by this option.

Default directory is `/var/opt/cray/dump`.

`dump_dir:` *directory*

Specifies the directory in which to save dumps.

Default directory is `/var/opt/cray/dump`.

`max_disk:` *nnnMB*`|unlimited`

Specifies the amount of disk space beyond which no new dumps will be created. This is not a hard limit; if `dumpd` sees that this directory has less than this amount of space, it starts a new dump, even if that dump subsequently uses enough space to exceed the `max_disk` limit.

The default value is `max_disk: unlimited`.

`no_space_action:` *action*

Specifies a command to be executed if the directory specified by the variable `dump_dir` does not have enough space free, as specified by `max_disk`.

Examples of possible actions:

Deletes the oldest dump in the dump directory:

```
no_space_action: rm -rf $dump_dir/$(ls -rt $dump_dir | head -1)
```

Moves the oldest dump somewhere useful:

```
no_space_action: mv $dump_dir/$(ls -t $dump_dir|head -1) /some/dump/archive
```

Sends E-mail to an administrator at admin@fictionalcraysite.com:

```
no_space_action: echo "" | mail -s "Not Enough Space in $dump_dir" \
admin@fictionalcraysite.com
```

3.22.3.3 Using the dumpd-dbadmin Tool

The dumpd daemon sits and waits for requests from NHC (or some other entity using the dumpd-request tool on the shared root.) When dumpd gets a request, it creates a database entry in the mznhc database for the request, and calls the script /opt/cray-xt-dumpd/default/bin/executor to read the dumpd.conf configuration file and perform the requested actions.

You can use the dumpd-dbadmin tool to view or delete entries in the mznhc database in a convenient manner.

3.22.3.4 Using the dumpd-request Tool

You can use the dumpd-request tool to send dump and reboot requests to dumpd from the SMW or the shared root.

A request includes a comma-separated list of actions to perform, and the node or nodes on which to perform the actions.

A typical request from NHC looks like this:

```
cname: c0-0c1s4n0 actions: halt,dump,reboot
```

You can define additional actions in the dumpd.conf configuration file; to use, you must execute the dumpd-request tool located on the shared root or the SMW. A typical call would be:

```
dumpd-request -a halt,dump,reboot -c c0-0c1s4n0
```

Or

```
dumpd-request -a myaction1,myaction2 -c c1-0c0s0n0,c1-0c0s0n1,c1-0c0s0n2,c1-0c0s0n3
```

For this example to work, you must define a myaction1 and myaction2 in the dumpd.conf file. See the examples in the configuration file for more detail.

3.23 Using `xtnmi` Command to Collect Debug Information From Hung Nodes

Caution: This is not a harmless tool to use to repeatedly get information from a node at various times; only use this command when you need debugging data from nodes that are in trouble. The `xtnmi` command output may be used to determine problems such as a core hang.

The sole purpose of the `xtnmi` command is to collect debug information from unresponsive nodes. As soon as that debug information is displayed to the console, the node panics.

For additional information, see the `xtnmi`(8) man page.

Monitoring System Activity [4]

4.1 Monitoring the System with the System Environmental Data Collector (SEDC)

To use the System Environmental Data Collector (SEDC) to collect data about internal cabinet temperatures, cooling system air pressures, critical voltages, etc., see *Using and Configuring System Environment Data Collections (SEDC)* (S–2491).

4.2 Displaying Installed SMW Release Level

Following a successful installation, the file `/opt/cray/hss/default/etc/smw-release` is populated with the installed SMW release level.

Example 36. Displaying installed SMW release level

```
% cat /opt/cray/hss/default/etc/smw-release
7.2.UP01
```

4.3 Displaying Current and Installed CLE Release Information

Following a successful installation, several files in the `/etc/opt/cray/release` directory are populated with various CLE release information.

The current `xtrelease` value (build number) is stored in `/etc/opt/cray/release/xtrelease`. The most recently installed CLE version number and update level is stored in `/etc/opt/cray/release/clerelease`.

Example 37. Displaying the current `xtrelease` value

```
crayadm@login:~> cat /etc/opt/cray/release/xtrelease
DEFAULT=5.2.14
```

The /etc/opt/cray/release/CLEinfo file contains a list of install-time options that other scripts may want access to, such as network type, installer version, and existence of the lustre file system.

Example 38. Displaying the most recently installed CLE release information

```
crayadm@login:~> cat /etc/opt/cray/release/CLEinfo
CLERELEASE=5.2.UP01
INSTALLERVERSION=b12
LUSTRE=yes
NETWORK=ari
XTRELEASE=5.2.14
```

Note: The CLEinfo file is an install-time "snapshot" and does not change; the release values may not be the currently booted version on your system.

4.4 Displaying Boot Configuration Information

Use the xtcli command to display the configuration information for the primary and backup boot nodes, the primary and backup SDB nodes, and the cpio path.

Procedure 13. Showing boot configuration information for the entire system

- To display boot configuration information for the entire system, execute the xtcli boot_cfg show command:

```
crayadm@smw:~> xtcli boot_cfg show
Network topology: class 2
=== xtcli_boot_cfg ===
[boot]: c0-0c0s0n1:ready,c0-0c0s0n1:ready
[sdb]: c1-0c0s1n1:ready
[cpio_path]: /tmp/boot/kernel.cpio_5.2.14-wGPFS
```

Procedure 14. Showing boot configuration information for a partition of a system

- To display boot configuration information for one partition in a system, specify the partition number, p*N*. p0 is always the whole system:

```
crayadm@smw:~> xtcli part_cfg show p1
```

4.5 Managing Log Files Using CLE and HSS Commands

Boot, diagnostic, and other Hardware Supervisory System (HSS) events are logged on the SMW in the /var/opt/cray/log directory, which is created during the installation process.

CLE logs are saved on the SMW in /var/opt/cray/log/*sessionid*.

Controller logs are saved on the SMW in `/var/opt/cray/log/controller/`*cabinet_name*`/`*controller_name*`/`*log_name*, where *cabinet_name* is of the form `c0-0`, `c1-0`, etc.; and where *controller_name* is of the form `c0-0` `c1-0`, etc. for the cabinet controller (CC) and is of the form `c0-0c0s0` for each blade controller (BC).

For more information, see the `intro_llm_logfiles`(5) man page.

4.5.1 Filtering the Event Log

The `xtlogfilter` command enables you to filter the event log for information such as the time a particular event occurred or messages from a particular cabinet.

Example 39. Finding information in the event log

To search for all console messages from node `c9-2c0s3n2`, type:

```
crayadm@smw:~> xtlogfilter -f
/var/opt/cray/log/event-yyyymmdd c9-2c0s3n2
```

For more information, see the `xtlogfilter`(8) man page.

4.5.2 Adding Entries to Log Files

You can add entries to the `syslog` with the `logger` command. For example, to identify the start or finish of system activities, use the `/bin/logger` command to log events into the system log. The message is then available to anyone who reads the log.

Example 40. Adding entries to `syslog` file

To mark the start of a new system test, type:

```
login# logger -is "Start of test 4A $(date) "
Start of test 4A Thu Jul 14 16:20:43 CDT 2011
```

The system log shows:

```
Jul 14 16:20:43 nid00003 xx[21332]:
Start of test 4A Thu Jul 13 16:20:43 CDT 2012
```

For more information, see the `logger`(1) man page.

4.5.3 Examining Log Files

Time-stamped log files of boot, diagnostic and other HSS events are located on the SMW in the `/var/opt/cray/log` directory. The time-stamped `bootinfo`, `console`, `consumer`, and `netwatch` log files are located in the `/var/opt/cray/log/`*sessionid* directory by default.

For example, the HSS `xtbootsys` command starts the `xtconsole` command, which redirects the output to a time-stamped log file, such as `/var/opt/cray/log/p0-20120716t104708/console-20120716`.

The `SMWinstall`, `SMWconfig`, and `SMWinstallCLE` commands create several detailed log files in the `/var/adm/cray/logs` directory. The log files are named using the PID of the `SMWinstall` or the `SMWinstallCLE` command; the exact names are displayed when the command is invoked.

4.5.4 Removing Old Log Files

The `xttrim` utility is used to provide a simple and configurable method to automate the compression and deletion of old log files. The `xttrim` utility is intended to be run on the SMW from `cron` and is automatically configured to do this as part of the Cray SMW software installation process. Review the `xttrim.conf` configuration file and ensure that `xttrim` will manage the desired directories and that the compression and deletion times are appropriate.

Note: The `xttrim` utility does not perform any action unless the `--confirm` flag is used to avoid unintended actions, nor will `xttrim` perform any action on open files. All actions are based on file-modified time.

For additional information, see the `xttrim`(8) and `xttrim.conf`(5) man pages.

4.6 Checking the Status of System Components

To check the status of the system or a component, use the `xtcli status` command on the SMW. By default, the `xtcli status` command returns the status of nodes.

Procedure 15. Showing the status of a component

- The `xtcli status` command has the form:

  `xtcli status [-n] [-m] [{-t` *type*`] [-a]}` *node_list*

  Note: The list should have component IDs only (no wild cards).

type may be: `cc`, `bc`, `cage`, `node`, `aries`, `aries_lcb`, `pdc`, or `qpdc`.

Use the `-m` option to display all nodes that were repurposed by using the `xtcli mark_node` command. (The `xtcli mark_node` command can be used to repurpose a service node to a compute role or to repurpose a compute node to a service role.)

Example 41. Display nodes that were repurposed with the `xtcli mark_node` command

`c0-0c0s2n0` is a service node, repurposed as a compute node. `c0-0c0s3n0` is a compute node, repurposed as a service node.

```
crayadm@smw:~> xtcli status -m c0-0c0
Network topology: class 2
Network type: Aries
              Nodeid: Service  Core Arch|  Comp state      [Flags]
   ----------------------------------------------------------------------
          c0-0c0s2n0:       -  SB16  X86|         off      [noflags|]
          c0-0c0s3n0: service  SB16  X86|         off      [noflags|]
   ----------------------------------------------------------------------
```

For more information, see the `xtcli`(8) man page.

4.7 Checking the Status of Compute Processors

To check that compute nodes are available after the system is booted, use the `xtprocadmin` command on a service node.

Example 42. Identifying nodes in down or admindown state

To identify if there are any nodes that are in a `down` or `admindown` state, execute the following command from a node:

```
nid00007:~> xtprocadmin | grep down
```

Example 43. Display current allocation and status of each compute processing element and the application that it is running

Use the user `xtnodestat` command to display the current allocation and status of

each compute processing element and the application that it is running. A simplified text display shows each processing element on the Cray system interconnection network. For example:

```
nid00007:~> xtnodestat
Current Allocation Status at Wed Jul 06 13:53:26 2011

     C0-0
  n3 AAaaaaaa
  n2 AAaaaaaa
  n1 Aeeaaaa-
c2n0 Aeeaaaaa
  n3 Acaaaaa-
  n2 cb-aaaa-
  n1 AA-aaaa-
c1n0 Aadaaaa-
  n3 SASaSa--
  n2 SbSaSa--
  n1 SaSaSa--
c0n0 SASaSa--
    s01234567

Legend:
   nonexistent node                  S  service node
;  free interactive compute node     -  free batch compute node
A  allocated interactive or ccm node ?  suspect compute node
W  waiting or non-running job        X  down compute node
Y  down or admindown service node    Z  admindown compute node

Available compute nodes:          0 interactive,         15 batch

Job ID     User       Size   Age        State           command line
--- ------ --------   -----  ---------  --------  ---------------------------------
a   3772974 user1   48    0h06m   run        app1
b   3773088 user2   2     0h01m   run        app2
c   3749113 user3   2     28h26m   run        app3
d   3773114 user4   1     0h00m   run        app4
e   3773112 user5   4     0h00m   run        app5
```

For more information, see the `xtprocadmin`(8) and `xtnodestat`(1) man pages.

4.8 Checking CNL Compute Node Connection

Use the Linux `ping` command to verify that a compute node is connected to the network. The Linux `ping` command must be run from a node, not run on the SMW.

Example 44. Verifying that a compute node is connected to the network

```
nid00007:~> ping nid00015
PING nid00015 (10.128.0.16) 56(84) bytes of data.
64 bytes from nid00015 (10.128.0.16): icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from nid00015 (10.128.0.16): icmp_seq=2 ttl=64 time=0.010 ms
```

For more information, see the Linux `ping`(8) man page.

4.9 Checking Link Control Block and Router Errors

The HSS `xtnetwatch` command monitors the Cray system interconnection network. It requests link control block (LCB) and router error information from the blade controller-based router daemons and specifies how often to sample for errors. It then detects events that contain the error information sent by these daemons and displays the information as formatted output in a log file.

You can specify which system components to sample and control the level of verbosity of the output, select the sampling interval, and log results to an output file.

Although the command can be invoked standalone from the SMW prompt, Cray recommends that you run `xtnetwatch` each time you boot the system with the `xtbootsys` command (the default). The output is a time-stamped log file such as:

```
/var/opt/cray/log/p0-20120803t185511/netwatch.p0-20120803t185511
```

Check the log file for fatal link errors and router errors. Fatal link errors signal faulty hardware. Fatal router errors can be generated either by hardware or software; they do not cause the network or individual links to become inoperable but imply that a single transfer was discarded.

Note: To turn off blade controller high-speed interconnect link monitoring, use the `xtnetwatch -d` option.

Example 45. Running `xtnetwatch` to monitor the system interconnection network

Sample the network once every 10 seconds using the least verbose display format:

```
crayadm@smw:~> xtnetwatch -i 10
120718 13:31:21  #################  #################  ##################   ##################
120718 13:31:21  LCB ID             Peer LCB           Category             Description
120718 13:31:21  #################  #################  ##################   ##################
Received all responses to request to start monitoring
Time of last linktune: Wed Jul 18 13:25:53 2012
Number of LCBs       : 112
Number of Blades     : 17
120718 13:31:22  c1-0c2s8a0l35     c1-0c2s0a0l32     ute                  Micropacket CRC \
                                                                          Error, cnt: 122
120718 13:31:22  c1-0c2s8a0l35     c1-0c2s0a0l32     ute
Sequential CRC Error
```

For more information, see the `xtnetwatch`(8) man page.

4.10 Displaying System Network Congestion Protection Information

Two utilities help you more easily identify the time and duration of system network congestion events, either by parsing through logs (`xtcpreport`) or in real time (`xtcptop`):

- The `xtcpreport` command uses information contained in the given `xtnlrd`

file to extract and display information related to system network congestion protection. Using this command, you can display a start time and an end time of the system network congestion protection information to display. See the `xtcpreport`(8) man page for additional information.

- The `xtcptop` command monitors an `xtnlrd` file that is currently being updated and displays real-time system network congestion protection information, including start time, duration, and apid. See the `xtcptop`(8) man page for additional information.

Note: You may need to execute the `module load congestion-tools` command to be able to call these utilities.

4.11 Monitoring the Health of PCIe Channels

Processors are connected to the high-speed interconnect network (HSN) ASIC through PCIe channels. The `xtpcimon` command is executed from the SMW and is started and run during the boot process. Any PCIe-related errors are reported to stdout, unless directed to a log file. `xtpcimon` also displays CLE-originated GHAL-based Advanced Error Reporting (AER) errors for PCIe. If the optional `/opt/cray/hss/default/etc/xtpcimon.ini` initialization file is present, the `xtpcimon` command uses the settings provided in the file.

For more information, see the `xtpcimon`(8) man page.

Example 46. Reporting PCIe-related errors to stdout

```
crayadm@smw:~> xtpcimon
starting
----> connection to event router made
121017 04:57:01  #############  #################   ##################
121017 04:57:01  Node           Category            Description
121017 04:57:01  #############  #################   ##################
Received all responses to request to start monitoring
121017 04:58:01  c0-0c0s7a0n1   CorrectableMemErr   0:0:0 AER Correctable: Non-fatal \
                                                       error (mask bit: 1)
121008 05:42:00  c0-0c1s6a0n2   CorrectableMemErr   Link CRC error (cnt: 3)
121008 05:43:30  c0-0c1s6a0n2   Info                Correctable/CRC error
```

4.12 Monitoring the Status of Jobs Started Under a Third-party Batch System

To monitor the status of jobs that were started under a third-party batch system, use the command appropriate to your batch system. For more information, see the documentation provided by your batch system vendor.

4.13 Using the cray_pam Module to Monitor Failed Login Attempts

The cray_pam module is a Pluggable Authentication Module (PAM). When configured, the cray_pam module provides information to the user at login time about any failed login attempts since their last successful login. See Using the cray_pam PAM to Log Failed Login Attempts on page 152 and the procedure to configure the cray_pam module, Procedure 30 on page 153.

4.14 Monitoring DDN RAID

Use Data Direct Networks tools to monitor DDN RAID. These can be accessed by telnetting to the RAID device from the SMW. To configure remote logging of DDN messages, see the *Installing Cray System Management Workstation (SMW) Software* (S–2480). For additional information, see your DDN documentation.

4.15 Monitoring NetApp, Inc. Engenio RAID

Use NetApp, Inc. Engenio tools to monitor NetApp, Inc. Engenio RAID. The NetApp, Inc. Engenio storage system uses SNMP to provide boot RAID messages. For additional information, see your NetApp, Inc. Engenio Storage System documentation.

4.16 Monitoring HSS Managers

This section provides procedures to view active sessions and to check whether the boot manager or the blade or cabinet controller daemons are running.

4.16.1 Examining Activity on the HSS Boot Manager

Use the HSS xtcli session show command to examine sessions in the boot manager. A session corresponds to running a specific command such as xtcli power up or xtcli boot. This command reports on sessions, not daemons.

Example 47. Looking at a session running on the boot manager

Execute the HSS xtcli session show command to view the session running on the boot manager:

```
crayadm@smw:~> xtcli session show BM
```

For more information about manager sessions, see the xtcli(8) man page.

4.16.2 Polling a Response from an HSS Daemon, Manager, or the Event Router

Use the HSS `xtalive` command to verify that an HSS daemon, manager, or the event router is responsive.

Example 48. Checking the boot manager

```
crayadm@smw:~> xtalive -l smw -a bm s0
```

For more information, see the `xtalive`(8) man page.

4.17 Validating the Health of the HSS

The `xtcheckhss` command initiates a series of tests that validate the health of the HSS. The tests gather and display information supplied by scripts located on blade controllers (BCs) and cabinet controllers (CCs). For a list of the `xtcheckhss` available tests, see the `xtcheckhss`(8) man page.

4.18 Monitoring Events

The HSS `xtconsumer` command enables you to monitor events mediated by the event router daemon `erd`, which runs passively.

Example 49. Monitoring for specific events

This command shows watching two events: `ec_heartbeat_stop`, which will be sent if either the node stops sending heartbeats or if the system interconnection network ASIC stops sending heartbeats, and `ec_l0_health`, which will be sent if any of the subcomponents of a blade controller report a bad health indication.

```
crayadm@smw:~> xtconsumer -b ec_heartbeat_stop ec_l0_health
```

Example 50. Checking events except heartbeat:

To display all events except heartbeats:

```
crayadm@smw:~> xtconsumer -x ec_l1_heartbeat
```

Use the `xthb` command to confirm the stopped heartbeat. Use the `xthb` command only when you are actively looking into a known problem because it is intrusive and degrades system performance.

For more information, see the `xtconsumer`(8) and `xthb`(8) man pages.

4.19 Monitoring Node Console Messages

The `xtbootsys` program will automatically start an `xtconsole` session which processes console messages for booted partition or system. The console messages will be written into `/var/opt/cray/log/`*sessionid*`/console-`*yyyymmdd* where the administrator may monitor them.

The `xtconsole` utility may only have one concurrent instance.

For more information, see the `xtconsole`(8) man page.

4.20 Showing the Component Alert, Warning, and Location History

Use the `xtcli comp_hist` command to display the component alert, warning, and location history. Either an error history, which displays alerts or warnings found on designated components, or a location history may be displayed.

Procedure 16. Displaying the location history for component `c0-0c0s0n1`

- Type:

```
crayadm@smw:~> xtcli comp_hist -o loc c0-0c0s0n1
```

For more information, see the `xtcli`(8) man page.

4.21 Displaying Component Information

Use the HSS `xtshow` command to identify compute and service components. Commands are typed as `xtshow --`*option_name*. You can also combine the `--service` or `--compute` option with other `xtshow` options to limit your selection to the specified type of node.

For a list of all `xtshow --`*option_name*s, see the `xtshow`(8) man page.

Example 51. Identifying all service nodes

```
crayadm@smw:~> xtshow --service
L1s ...
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
         c0-0c0s1: service        X86|       ready      [noflags|]
         c1-0c0s0: service        X86|       ready      [noflags|]
         c1-0c0s1: service        X86|       ready      [noflags|]
         c2-0c0s1: service        X86|       ready      [noflags|]
         c2-0c1s1: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n0: service        X86|       empty      [noflags|]
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n3: service        X86|       empty      [noflags|]
```

```
       c0-0c0s1n0: service        X86|       empty      [noflags|]
       c0-0c0s1n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s1n2: service  SB08  X86|       ready      [noflags|]
       c0-0c0s1n3: service        X86|       empty      [noflags|]
       c1-0c0s0n0: service        X86|       empty      [noflags|]
       c1-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c1-0c0s0n2: service  SB08  X86|       ready      [noflags|]
       c1-0c0s0n3: service        X86|       empty      [noflags|]
       c1-0c0s1n0: service        X86|       empty      [noflags|]
       c1-0c0s1n1: service  SB08  X86|       ready      [noflags|]
       c1-0c0s1n2: service  SB08  X86|       ready      [noflags|]
       c1-0c0s1n3: service        X86|       empty      [noflags|]
   .
   .
   .
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
       c0-0c0s1a0: service        X86|          on      [noflags|]
       c1-0c0s0a0: service        X86|          on      [noflags|]
       c1-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c1s1a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
    c0-0c0s0a0l07: service        X86|          on      [noflags|]
    c0-0c0s0a0l10: service        X86|          on      [noflags|]
    c0-0c0s0a0l11: service        X86|          on      [noflags|]
    c0-0c0s0a0l12: service        X86|          on      [noflags|]
    c0-0c0s0a0l13: service        X86|          on      [noflags|]
    c0-0c0s0a0l14: service        X86|          on      [noflags|]
    c0-0c0s0a0l15: service        X86|          on      [noflags|]
    c0-0c0s0a0l16: service        X86|          on      [noflags|]
  .
  .
  .
```

Example 52. Showing compute nodes in the `disabled` state

```
crayadm@smw:~> xtshow --compute --disabled
L1s ...
Cages ...
L0s ...
Nodes ...
       c0-0c2s0n3:       -        X86|    disabled      [noflags|]
      c0-0c2s11n0:       -        X86|    disabled      [noflags|]
      c0-0c2s11n3:       -        X86|    disabled      [noflags|]
      c1-0c0s11n2:       -        X86|    disabled      [noflags|]
Aries ...
AriesLcbs ...
```

Example 53. Showing components with a status of not empty

```
crayadm@smw:~> xtshow --not_empty c0-0c0s0
L1s ...
            c0-0:       -           |          on      [warn|alert|]
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
    c0-0c0s0a0l07: service        X86|          on      [noflags|]
    c0-0c0s0a0l10: service        X86|          on      [noflags|]
    c0-0c0s0a0l11: service        X86|          on      [noflags|]
    c0-0c0s0a0l12: service        X86|          on      [noflags|]
    c0-0c0s0a0l13: service        X86|          on      [noflags|]
    c0-0c0s0a0l14: service        X86|          on      [noflags|]
    c0-0c0s0a0l15: service        X86|          on      [noflags|]
    c0-0c0s0a0l16: service        X86|          on      [noflags|]
    c0-0c0s0a0l17: service        X86|          on      [noflags|]
    c0-0c0s0a0l20: service        X86|          on      [noflags|]
    c0-0c0s0a0l21: service        X86|          on      [noflags|]
    c0-0c0s0a0l22: service        X86|          on      [noflags|]
    c0-0c0s0a0l23: service        X86|          on      [noflags|]
    c0-0c0s0a0l24: service        X86|          on      [noflags|]
    c0-0c0s0a0l25: service        X86|          on      [noflags|]
    c0-0c0s0a0l26: service        X86|          on      [noflags|]
    c0-0c0s0a0l27: service        X86|          on      [noflags|]
   .
   .
   .
```

4.22 Displaying Alerts and Warnings

Use the `xtshow` command to display alerts and warnings. Type commands as **`xtshow`** `--`*option_name*, where *option_name* is `alert`, `warn`, or `noflags`.

Note: Alerts are not propagated through the system hierarchy, so you only receive information for the component you are examining. A node can have an alert, but you would not see it if you ran the `xtshow --alert` command for a cabinet. In a similar fashion, you would not detect an alert on a cabinet if you checked the status of a node.

Example 54. Show all alerts on the system

```
smw:~> xtshow --alert
```

For additional information, see the `xtshow`(8) man page.

Alerts and warnings typically occur while the HSS `xtcli` command operates; these alerts and warnings are listed in the command output with an error message. After they are generated, alerts and warnings become part of the state for the component and remain set until you manually clear them. For example, the temporary loss of a heartbeat by the blade controller may set a warning state on a chip.

4.23 Clearing Flags

Use the `xtclear` command to clear system information for components you select. Type commands as `xtclear --`*option_name*, where *option_name* is `alert`, `reserve`, or `warn`.

Example 55. Clear all warnings in specified cabinet

To clear all warnings in cabinet `c13-2`:

```
smw:~> xtclear --warn c13-2
```

You must clear alerts, reserves, and warnings before a component can operate. Clearing an alert on a component frees its state so that subsequent commands can execute (see Appendix B, System States on page 393).

For more information, see the `xtclear`(8) man page.

4.24 Displaying Error Codes

When a Hardware Supervisory System (HSS) event error occurs, the related message is displayed on the SMW. You can also use the `xterrorcode` command on the SMW to display a single error code or the entire list of error codes.

Example 56. Displaying HSS error codes

```
smw:~>  xterrorcode errorcode
```

A system error code entered in a log file is a bit mask; invoking the `xterrorcode` *bitmask_code_number* command on the SMW displays the associated error code.

Example 57. Displaying an HSS error code using its bit mask number

```
smw:~> xterrorcode 131279
Maximum error code (RS_NUM_ERR_CODE) is 297
code = 207, string = 'node voltage fault'
```

Managing User Access [5]

For a description of administrator accounts that enable you to access the functions described in this chapter, see Administering Accounts on page 117.

5.1 Load Balancing Across Login Nodes

Having all users log on to the same login node may overload the node. (Also, see the Caution in Login Nodes on page 38.) For typical interactive usage, a single login node is expected to handle 20 to 30 batch users or 20 to 40 interactive users with double this number of user processes. You can use the `lbnamed` load-balancing software to distribute logins to different login nodes. The `lbnamed` daemon is a name server that gathers the output of `lbcd` client daemons to select the least loaded node, provides DNS-like responses, interacts with the corporate DNS server, and directs the user login request to the least busy login node.

Because `lbnamed` runs on the SMW, `eth0` on the SMW must be connected to the same network from which users log on the login nodes.

> **Note:** If security considerations do not allow you to put the SMW on the public network, `lbnamed` may be installed on an external server. This can be any type of computer running the SUSE Linux Enterprise Server (SLES) operating system (not a 32-bit system). However, this option is not a tested or supported Cray configuration.

The behavior of the `lbnamed` daemon is site-configurable and determined by the contents of the `/etc/opt/cray-xt-lbnamed/lbnamed.conf` and `/etc/opt/cray-xt-lbnamed/poller.conf` configuration files. For details about configuring the load balancer, see Configuring the Load Balancer on page 159, and the `lbcd`(8), `lbnamed`(8), and `lbnamed.conf`(5) man pages.

5.2 Passwords

The default passwords for the `root` and `crayadm` accounts are the same for the System Management Workstation (SMW), the boot node, and the shared root.

Default passwords for the `root` and `crayadm` accounts are provided in the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444). Also, default MySQL passwords and an example of how to change them are provided in the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444). Cray recommends changing these default passwords as part of the software installation process.

5.2.1 Change the Default SMW Passwords

After completing the installation, change the default SMW passwords. The SMW contains its own `/etc/passwd` file that is separate from the password file for the rest of the system. To change the passwords on the SMW, log on to the SMW as `root` and execute the following commands:

```
crayadm@smw:~> su - root
smw:~# passwd root
smw:~# passwd crayadm
smw:~# passwd cray-vnc
smw:~# passwd mysql
```

To change the default iDRAC6 password see Procedure 124 on page 428.

5.2.2 Changing `root` and `crayadm` Passwords on Boot and Service Nodes

For security purposes, it is desirable to change the passwords for the `root` and `crayadm` accounts on a regular basis.

Use the Linux `passwd` command to change the `/etc/passwd` file. For information about using the `passwd` command, see the `passwd`(1) man page.

Procedure 17. Changing the `root` and `crayadm` passwords on boot and service nodes

1. The boot node contains its own `/etc/passwd` file that is separate from the password file for the rest of the system. To change the passwords on the boot node, use these commands. You will be prompted to type and confirm new root and administrative passwords.

   ```
   boot:~ # passwd root
   boot:~ # passwd crayadm
   ```

2. To change the passwords on the other service nodes, you must run these commands on the shared root. Again, you will be prompted to type and confirm new passwords for the `root` and `crayadm` accounts.

   Note: If the SDB node is not started, you must add the `-x /etc/opt/cray/sdb/node_classes` option to the `xtopview` command in this procedure.

```
boot:~ # xtopview
default/:/ # passwd root
default/:/ # passwd crayadm
default/:/ # exit
```

For more information about using the `xtopview` command, see Managing System Configuration with the `xtopview` Tool on page 133, and the `xtopview`(8) man page.

5.2.3 Changing the `root` Password on CNL Compute Nodes

Procedure 18. Changing the `root` password on CNL compute nodes

For compute nodes, update the `root` account password in the `/opt/xt-images/templates/default/etc/shadow` file on the SMW.

Note: To make these changes for a system partition, rather than for the entire system, replace `/opt/xt-images/templates` with `/opt/xt-images/templates-p`*N*, where *N* is the partition number.

1. Copy the master password file to the template directory.

```
smw:~ # cp /opt/xt-images/master/default/etc/shadow \
/opt/xt-images/templates/default/etc/shadow
```

2. Edit the password file to include a new encrypted password for the `root` account.

```
smw:~ # vi /opt/xt-images/templates/default/etc/shadow
```

3. After making these changes, update the boot image by following the steps in Procedure 2 on page 64.

5.2.4 Changing the HSS Data Store (MySQL) Password

Use the `hssds_init` command to change the HSS data store (MySQL) `root` password. The `hssds_init` command prompts you for the current HSS data store (MySQL) `root` password. When you type the current and new passwords, they are not echoed.

Note: The `hssds_init` utility is run by the `SMWinstall` command, so you do not have to run it during installation of an SMW release package.

For additional information, see the `hssds_init`(8) man page.

5.2.5 Changing Default MySQL Passwords on the SDB

Procedure 19. Changing default MySQL passwords on the SDB

For security, you should change the default passwords for MySQL database accounts. The valid characters for use in MySQL passwords are:

```
 ! " # $ % & ' ( )
* + , - . / 0 1 2 3
4 5 6 7 8 9 : ; < =
> ? @ A B C D E F G
H I J K L M N O P Q
R S T U V W X Y Z [
\ ] ^ _ ` a b c d e
f g h i j k l m n o
p q r s t u v w x y
z { | } ~
```

1. If you have not set a site-specific MySQL password for root, type the following commands. Press the `Enter` key when prompted for a password.

```
boot:~ # ssh root@sdb
sdb:~ # mysql -h localhost -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.31-log Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> set password for 'root'@'localhost' = password('newpassword');
Query OK, 0 rows affected (0.00 sec)
mysql> set password for 'root'@'%' = password('newpassword');
Query OK, 0 rows affected (0.00 sec)
mysql> set password for 'root'@'sdb' = password('newpassword');
Query OK, 0 rows affected (0.00 sec)
```

2. (Optional) Set a site-specific password for other MySQL database accounts.

   a. To change the password for the `sys_mgmt` account, type the following MySQL command. You must also update `.my.cnf` in step 4.

```
mysql> set password for 'sys_mgmt'@'%' = password('newpassword');
Query OK, 0 rows affected (0.00 sec)
```

b. To change the password for the `basic` account, type the following MySQL command. You must also update `/etc/opt/cray/sysadm/odbc.ini` in step 5.

Note: Changing the password for the `basic` MySQL user account will not provide any added security. This read-only account is used by the system to allow all users to run `xtprocadmin`, `xtnodestat`, and other commands that require SDB access.

```
mysql> set password for 'basic'@'%' = password('newpassword');
Query OK, 0 rows affected (0.00 sec)
```

Note: When making changes to the MySQL database, your connection may time out; however, it is automatically reconnected. If this happens, you will see messages similar to the following. These messages may be ignored.

```
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    21127
Current database: *** NONE ***

Query OK, 0 rows affected (0.00 sec)
```

3. Exit from MySQL and the SDB.

```
mysql> exit
Bye
sdb:~ # exit
boot:~ #
```

4. (Optional) If you set a site-specific password for `sys_mgmt` in step 2, update the `.my.cnf` file for root with the new password. Additionally, update the `.odbc.ini.root` file with the new password.

a. Edit `.my.cnf` for root on the boot node.

```
boot:~ # vi /root/.my.cnf
[client]
user=sys_mgmt
password=newpassword
```

b. Edit `.my.cnf` for root in the shared root.

```
boot:~ # xtopview
default/:/ # vi /root/.my.cnf
[client]
user=sys_mgmt
password=newpassword
default/:/ # exit
boot:~ #
```

c. Edit `.odbc.ini.root` for the root on the boot node. Update **each** database section with the new password.

```
boot:~ # vi /root/.odbc.ini.root
Driver          = MySQL_ODBC
Description     = Connector/ODBC Driver DSN
USER            = sys_mgmt
PASSWORD        = newpassword
```

After updating the file, copy `.odbc.ini.root` to `.odbc.ini`.

```
boot:~ # cp –p /root/.odbc.ini.root /root/.odbc.ini
```

d. Edit `.odbc.ini.root` for the root in the shared root. Update each database section with the new password.

```
boot:~ # xtopview
default/:/ # vi /root/.odbc.ini.root
Driver          = MySQL_ODBC
Description     = Connector/ODBC Driver DSN
USER            = sys_mgmt
PASSWORD        = newpassword
default/:/ # exit
boot:~ #
```

After updating the file, restart the SDB service on all service nodes.

```
boot:~ # pdsh -a /etc/init.d/sdb restart
```

5. (Optional) If you set a site-specific password for `basic` in step 2, update **each** datasource name in the `/etc/opt/cray/sysadm/odbc.ini` file with the new password. Additionally, update the `/root/.odbc.ini` file with the new password.

a. Edit `/etc/opt/cray/sysadm/odbc.ini` for the basic user on the boot node. Update each database section with the new password.

```
boot:~ # vi /etc/opt/cray/sysadm/odbc.ini
Driver          = MySQL_ODBC
Description     = Connector/ODBC Driver DSN
USER            = basic
PASSWORD        = newpassword
```

b. Edit `/root/odbc.ini` in the shared root. Update each database section with the new password.

```
boot:~ # xtopview
default/:/ # vi /root/odbc.ini
Driver          = MySQL_ODBC
Description     = Connector/ODBC Driver DSN
USER            = basic
PASSWORD        = newpassword
default/:/ # exit
boot:~ #
```

5.2.6 Assigning and Changing User Passwords

Because a Cray system has a read-only shared-root configuration, users cannot execute the `passwd` command on a Cray system to change their password. If your site has an external authentication service such as Kerberos or LDAP, users should follow your site instructions to update their passwords. If your site does not have external authentication set up, you can implement a manual mechanism, such as having users change their password on an external system and you periodically copying their entries in the external `/etc/passwd`, `/etc/shadow`, and `/etc/group` files to the equivalent Cray system files in the default `xtopview`.

Warning: Be careful to not overwrite Cray system accounts (`crayadm`, `cray_vnc` and standard Linux accounts such as `root`) in the `/etc/passwd`, `/etc/shadow`, and `/etc/group` files.

5.2.7 Logins That Do Not Require Passwords

All logins must have passwords; however, you can set up a passwordless `ssh` by creating an `ssh` key with a null passphrase and distributing that `ssh` key to another computer.

While the key-based authentication systems such as OpenSSH are relatively secure, convenience and security are often mutually exclusive. Setting up passphrase-less `ssh` is convenient, but the security ramifications can be dire; if the local host is compromised, access to the remote host will be compromised as well.

If you wish to use passphrase-less authentication, Cray encourages you to consider using `ssh-agent` if available, or take other steps to mitigate risk.

5.3 Administering Accounts

Your Cray system supports several types of accounts:

- Boot node accounts allow only system administrator (`crayadm`) and superuser (`root`) access. To modify configuration files, the administrator must become superuser by supplying the `root` account password.
- SMW user account access is managed using local files (that is, `/etc/passwd`, `/etc/shadow`, etc.). In addition to the standard Linux system accounts, Cray includes an account named `crayadm`; `crayadm` is used for many of the Cray system management functions, such as booting the system.
- Cray provides a Virtual Network Computing (VNC) account on the SMW (for details, see Appendix C, Remote Access to the SMW on page 395).

5.3.1 Managing Boot Node Accounts

The only accounts that are supported on the boot node are `root` (superuser), `crayadm` (administrator), and those accounts for various services such as network time protocol (NTP). The boot node does not support user accounts.

The boot node has an `/etc/passwd` file that is separate from the password file for the rest of the system. For a list of default passwords, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

5.3.2 Managing User Accounts That Must Be Maintained on the Cray System Directly

Note: Normally, user account and passwords are managed through an external LDAP or Kerberos server. However, for those accounts that must be maintained on the Cray system directly, in case LDAP or Kerberos service are not available, this section describes how you manage them in shared root.

User accounts are set up on the shared-root file system by using the `xtopview` command. Your Cray system supports 16-bit and 32-bit user IDs (UIDs). The 16-bit user IDs run 0-65535; that is 0-(2^{16}-1). The 32-bit user IDs run 0-(2^{32}-1), although Cray systems are limited to a maximum of 65,536 user accounts, including those that are predefined, such as `root`, `crayadm`, and `mysql`.

For more information about using the `xtopview` command in the default view, see Managing System Configuration with the `xtopview` Tool on page 133, and the `xtopview`(8) man page. For more information about `mysql` accounts, see Database Security on page 193.

5.3.2.1 Adding a User or Group

To add additional accounts to the shared root for login nodes, use the `groupadd` and `useradd` commands using the `xtopview` command in the default view.

Example 58. Adding a group

To add the group `xtusers` with a *gid* of `5605`, type:

```
boot:~ # xtopview
default/:/ # groupadd -g 5605 xtusers
default/:/ # exit
```

The above `groupadd` command adds group `xtusers` to `/etc/group`.

Example 59. Adding a user account

This example creates a new user `bobp` from `xtopview` in the default view. The

new user account, bobp, has a user ID of 12645, a home directory bobp, and runs a /bin/bash login shell. Then, as root, create the user's home directory and chown the directory to the new user.

```
boot:~ # xtopview
default/:/ # useradd -d /home/users/bobp -g 5605 -s /bin/bash -u 12645 bobp
default/:/ # exit
boot:~ # ssh root@login
login:~ # mkdir -p /home/users/bobp
login:~ # chown -R bobp:xtusers /home/users/bobp
```

After the account is created, use the passwd command to set a password in either /etc/passwd or /etc/shadow.

For more information, see the useradd(8), passwd(1), and groupadd(8) man pages.

5.3.2.2 Removing a User or Group

To remove a user account, first remove all files, jobs, and other references to the user. Then using the xtopview command in the default view, remove users or groups by using Linux commands /usr/sbin/userdel and /usr/sbin/groupdel, respectively; and, as root, remove the user's home directory.

Example 60. Removing a user account

To remove the user bobp and the user's home directory, type:

```
boot:~ # xtopview
default/:/ # userdel -r bobp
default/:/ # exit
boot:~ # ssh root@login
login:~ # rm -rf /home/users/bobp
login:~ # exit
```

For more information, see the userdel(8) and groupdel(8) man pages.

5.3.2.3 Changing User or Group Information

To change user and group information, use Linux commands. For more information, see the usermod(8) and groupmod(8) man pages.

5.3.2.4 Assigning Groups of CNL Compute Nodes to a User Group

Use the /etc/opt/cray/sdb/attr.defaults file label attribute to assign groups of CNL compute nodes to specific user groups without the need to partition the system. For more information, see Setting Node Attributes Using the /etc/opt/cray/sdb/attr.xthwinv.xml and /etc/opt/cray/sdb/attr.defaults Files on page 202.

5.3.2.5 Associating Users with Projects

You can assign project names for users to submit jobs in order to determine project charges. Project names can be up to 80 characters long.

To associate users with project names, add the following line to their individual login scripts in their home directories:

```
set_account a_project_name
```

After accounts are set, users do not have to manually run the `set_account` command at each login.

If your users run batch jobs, they can set a project code; for example, when using PBS Professional, a user can set a project code with the `ENVIRONMENT` variable. This associates the project code with the job in the accounting database. For more information, see the documentation provided by your batch system vendor.

5.3.2.6 Enabling LDAP Support for User Authentication

To enable LDAP support for user authentication, you must edit files as shown, in addition to making any other standard LDAP configuration setting changes necessary for your site.

Note: The following changes should be made using `xtopview` in the default view.

```
boot:~ # xtopview
default/:/ # vi /etc/pam.d/common-account-pc
```

In file `/etc/pam.d/common-account`, replace:

```
account required        pam_unix2.so
```

with

```
account sufficient      pam_ldap.so config=/etc/openldap/ldap.conf
account required        pam_unix2.so
```

```
default/:/ # vi /etc/pam.d/common-auth-pc
```

In file `/etc/pam.d/common-auth`, replace:

```
auth    required        pam_env.so
auth    required        pam_unix2.so
```

with

```
auth    required        pam_env.so
auth    sufficient      pam_ldap.so config=/etc/openldap/ldap.conf
auth    required        pam_unix2.so
```

```
default/:/ # vi /etc/pam.d/common-password-pc
```

In file /etc/pam.d/common-password, replace:

```
password required       pam_pwcheck.so  nullok
password required       pam_unix2.so    nullok use_authtok
```

with

```
password required       pam_pwcheck.so  nullok
password sufficient     pam_ldap.so config=/etc/openldap/ldap.conf
password required       pam_unix2.so    nullok use_authtok
```

```
default/:/ # vi /etc/pam.d/common-session-pc
```

In file /etc/pam.d/common-session, replace:

```
session required        pam_limits.so
session required        pam_unix2.so
session optional        pam_umask.so
```

with

```
session required        pam_limits.so
session sufficient      pam_ldap.so config=/etc/openldap/ldap.conf
session required        pam_unix2.so
session optional        pam_umask.so
```

Create the /var/run/slapd directory, and set its ownership to the ldap user and group.

```
default/:/ # mkdir /var/run/slapd/
default/:/ # chown ldap:ldap /var/run/slapd/
```

On the boot root, in the xtopview default view, make your site-specific changes to the /etc/openldap/ldap.conf or /etc/ldap.conf, /etc/nsswitch.conf, /etc/sysconfig/ldap, /etc/passwd, and /etc/group files.

Note: Adding the LDAP servers to the local host file allows you to **not** run DNS on the SDB and MDS. The SDB and MDS need access to the LDAP server. You can set up this access through RSIP or NAT; see Configuring Realm-specific IP Addressing (RSIP) on page 209.

```
default/:/ # exit
```

5.3.3 Setting Disk Quotas for a User on the Cray Local, Non-Lustre File System

The `quota` and `quota-nfs` RPMs are installed by default. You can activate disk quotas for a user on service nodes on the Cray local, non-Lustre file system. You must activate two boot scripts, as discussed in the `README.SUSE` file located in `/usr/share/doc/packages/quota`.

Note: When following the procedure in the `README.SUSE` file, remember that any commands should be issued from within the default view of `xtopview`. Also, use the `chkconfig` command instead of the `yast2` run level editor to turn on `quota` and `quotad` services:

```
boot:~ # xtopview
default/:/ # chkconfig boot.quota on
default/:/ # chkconfig quotad on
default/:/ # exit
```

Then start the services on all service nodes; either reboot the system or execute `/etc/init.d/boot.quota start; /etc/init.d/quotad start` on each service node.

After the quota services have been enabled, for each user you can use standard Linux quota commands to do the following:

- Enable quotas (`quotaon` command)
- Check quotas (`quotacheck` command)
- Set quotas (`edquota` command)

When a quota is exceeded, the quotas subsystem warns users when they exceed their allotted limit, but it allows some extra space for current work (that is, there is a hard limit and a soft limit).

For more information, see the `quotaon`(8), `quotacheck`(8), and `edquota`(8) Linux man pages.

5.4 About Modules and Modulefiles

The Modules software package enables your users to modify their environment dynamically by using *modulefiles*. The `module` command is a user interface to the Modules package. The `module` command interprets modulefiles, which contain Tool Command Language (Tcl) code, and dynamically modifies shell environment variables such as `PATH`, and `MANPATH`.

For more information about the Modules software package, see the `module`(1) and `modulefile`(4) man pages.

The shell configuration files `/etc/csh.cshrc.local` and `/etc/bash.bashrc.local` contain module commands which establish the default user environment which is set by the system at login time.

To support customer-specific needs, you can create your own modulefiles for a product set for your users; for details, see Appendix E, Creating Modulefiles on page 405.

5.5 About the /etc/*rc.local Files

The /etc/csh/cshrc.local and /etc/bash/bashrc.local files contain several ordered blocks, each clearly delimited by ##BEGIN and ##END tags.

The CLE installation and upgrade process creates and maintains the first two ##BEGIN and ##END blocks. These blocks contain clearly delimited sections for operating system and programming environment changes only. These sections should not be modified by the administrator.

The administrator should add site local changes within the SITE-set-up block only. A CLE upgrade may modify the operating system blocks in place, preserving SITE-set-up local changes you have made.

5.6 System-wide Default Modulefiles

The /etc/csh/cshrc.local and /etc/bash/bashrc.local files load Base-opts, which loads two lists of modulefiles: a default list and a site-specified local list.

The default list differs between the SMW and the Cray system. On the SMW, the file /etc/opt/cray/modules/Base-opts.default.SMW contains the list of the CLE modulefiles to load by default. On the Cray system, the file /etc/opt/cray/modules/Base-opts.default contains the list of CLE modulefiles to load by default.

Additionally, all the modulefiles listed in the file /etc/opt/cray/modules/Base-opts.default.local are loaded. Edit this file to make your site-specific changes.

The /etc/opt/cray/modules/Base-opts.default.local file initially includes the admin-modules modulefile, which loads a full set of modulefiles. You do not need to manually load the admin-modules modulefile, unless the you have removed it from the default list. The CLE installation process removes admin-modules modulefile from the default list on login nodes.

The files on the Cray system are installed on both the boot root and the shared root.

An example file, /etc/opt/cray/modules/Base-opts.default.local.example, is also provided. The example file is a copy of the /etc/opt/cray/modules/Base-opts.default.local file provided for an initial installation.

5.7 Configuring the Default Programming Environment (PE)

The Cray, PGI, GCC, PathScale (Cray XE systems only), and Intel compilers are available to Cray System users, if installed. A Programming Environment is comprised of a compiler and its supporting libraries and tools.

The system wide default PE is set to `PrgEnv-cray` in the `PE-set-up` block in the `/etc/*rc.local` files. For Cray XE sites without a Cray Compiling Environment license, `PrgEnv-pgi` is the default. If you wish to alter the system default PE, do not edit the `PE-set-up` block manually.

To change the default PE and add more PE user defaults, add appropriate instructions to the `SITE-set-up` block. The instructions in the `SITE-set-up` block are not altered by operating system installations. The instructions are evaluated after the `PE-set-up` block, so make sure new instructions do not conflict with the ones in the `PE-set-up` block.

For example, if you want to change the default PE to `PrgEnv-abc`, add this to the `SITE-set-up` block in `/etc/*rc.local` files:

```
module unload PrgEnv-cray
module load PrgEnv-abc
```

Targeting modules are released in the `xt-asyncpe` and the `craype` packages and installed in either the `/opt/cray/xt-asyncpe/default/modulefiles` or `/opt/cray/craype/default/modulefiles` directory, depending on the package name.

The following commands display the list of available targeting modulefiles:

```
module avail xt-asyncpe
module avail craype
```

If there are no targeting modules loaded in the user's environment, the compiler driver scripts (`cc`, `CC`, `ftn`) set the CPU target to `sandybridge` on Cray XC30 systems, and to `interlagos` on Cray XE and Cray XK systems. To change the default CPU target, configure `/etc/*rc.local` to load the appropriate `craype-*` target module.

For example, to set the default target to *xyz* in the `*rc.local` files, add the following line to the `SITE-set-up` section:

```
module add craype-xyz
```

Other commonly used modules and settings in the `SITE-set-up` block are:

```
#load a workload manager
module load pbs
#define target architecture
module load craype-abudhabi
#mpich2 is not loaded by PrgEnv-cray
module load cray-mpich2
#enable abnormal termination processing (see intro_atp)
setenv ATP_ENABLED 1
```

5.8 Using the pam_listfile Module in the Shared Root Environment

The Linux pam_listfile Pluggable Authentication Module (PAM) may be used to maintain a list of authorized users. Using the pam_listfile PAM may also help to reduce impacts on service nodes if users consume too many resources (see Caution in Login Nodes on page 38).

The pam_listfile PAM requires that the file specified with the file= parameter be a regular file. The usual approach of storing the file in the /etc directory does not work in the shared-root environment of Cray systems: files in the /etc directory are symbolic links, so the required file must be created in a directory other than the /etc directory. For example, you can place it in persistent /var or another directory that is not controlled by the shared root.

Example 61. Creating a pam_listfile list file

This example assumes you have created an empty pam_listfile called /var*/path/to*/pam_listfile_authorized_users_list. It adds authorized users to it.

```
boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -c login
class/login/:# vi /var/path/to/pam_listfile_authorized_users_list

user1
user2
...
```

Example 62. Adding a line to /etc/pam.d/sshd to enable pam_listfile

Edit the pam.d/sshd file to include an alternative path for file=.

```
class/login/:#  vi /etc/pam.d/sshd

auth required pam_listfile.so \
file=/var/path/to/pam_listfile_authorized_users_list
```

If you need nodes to have different pam_listfile list files, create the list files and specialize the PAM configuration files (such as pam.d/sshd) to point to them.

5.9 ulimit Stack Size Limit

The login environment defaults to the kernel default stack size limit. To set up the default user environment to have an unlimited stack size resource limit, add the following to /etc/profile.local in the shared root:

```
ulimit -Ss unlimited
```

5.10 Stopping a User's Job

This section describes how to stop a user's job.

5.10.1 Stopping a Job Running in Interactive Mode

If the job is running on a CNL compute node in interactive mode (through `aprun`), perform the following procedure.

Procedure 20. Stopping a job running in interactive mode

- Use the `apkill -`*signal apid* command to send a signal to all processes that are part of the specified application (*apid*); signal 15 (`SIGTERM`) is sent by default.

The signaled application must belong to the current user unless the user is a privileged user. For more information, see the `aprun`(1) and `apkill`(1) man pages.

5.10.2 Stopping a Job Running Under a Batch System

To stop a job that is running under a batch system, see the documentation provided by your batch system vendor.

Example 63. Stopping a job running under PBS Professional

If the job is running under PBS Professional, use the `qdel` command and name the job.

To terminate job 104, type:

```
% qdel 104
```

For Platform LSF the command to kill a job is `bkill`.

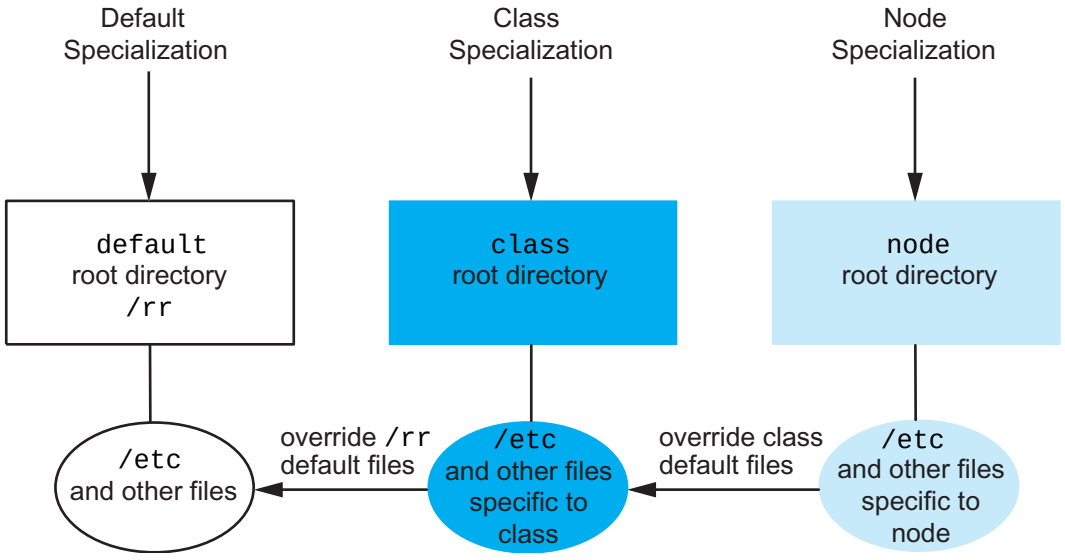
6.1 Configuring the Shared-root File System on Service Nodes

CLE implements a shared-root file system where `/` is exported from the boot node and is mounted as read-only on all service nodes. To overcome the restriction that all nodes must have the same shared-root file system, `/etc` directories can be symbolic links to unique directories that have the same structure as the default `/etc` directory but contain modified files. These node-specific files reside in subdirectories in the `/.shared/base` directory.

Specialization is the process of changing the link to a file in the `/etc` directory to point to a unique file for one, a few, or all nodes. You can specialize one or more files for an individual node or for a class (type) of nodes, such as login. You must be `root` user to configure the shared-root file system in this manner. You can specialize files when you install the system or at a later time.

The hierarchical structure of the specialized files is shown in Figure 2. Node specialization is more specific than class specialization. Class specialization is more specific than default specialization. Generally, about 98% of what the service nodes use is the default version of the shared root.

Figure 2. Types of Specialization



6.1.1 Specialization

You specialize files when you need to point to a unique version of a file in the `/etc` directory rather than to the standard version of the file that is shared on all nodes. For example, you might specialize files when differences exist in hardware, network configuration, or boot scripts or when there are services that run on a single node. You can also specialize files for a class of nodes that have a particular function, such as login.

Generally, files are specialized as part of the installation process, but the process can be done at any time. It is good practice to enter the `xtopview` shell (see Managing System Configuration with the `xtopview` Tool on page 133) and then specialize your files (see Specializing Files on page 136).

Table 2 lists files and directories that you can specialize by class and the reasons to do so. Table 3 lists files and directories that you can specialize by node and the reasons to do so. In these tables, * refers to "wildcard" characters that represent no characters or any number of characters.

Table 2. File Specialization by Class

File or Directory	Reason for Specialization
`/etc/auditd.conf`	Cray Audit configured on login nodes.
`/etc/audit.rules`	Cray Audit configured on login nodes.
`/etc/cron*`	Different classes need custom crontabs.
`/etc/fstab`	I/O nodes need to mount other file systems.
`/etc/hosts.{allow,deny}`	Must restrict logins on login nodes.
`/etc/init.d/boot.d/*`	Different classes have different start-up scripts enabled.
`/etc/init.d/rc*/`	Different classes have different start-up scripts enabled.
`/etc/issue`	Different classes have different messages.
`/etc/modprobe.conf`	I/O and login nodes have different hardware.
`/etc/motd`	Different classes have different messages.
`/etc/pam*`	Authentication is class-specific.
`/etc/profile.d/*`	Login nodes have custom environments.
`/etc/resolv.conf`	Hosts that interact with external servers need special resolver configurations.
`/etc/security/*`	Authorization and system limits are class-specific.
`/etc/sysconfig/network/*`	I/O and login nodes need custom network configuration.

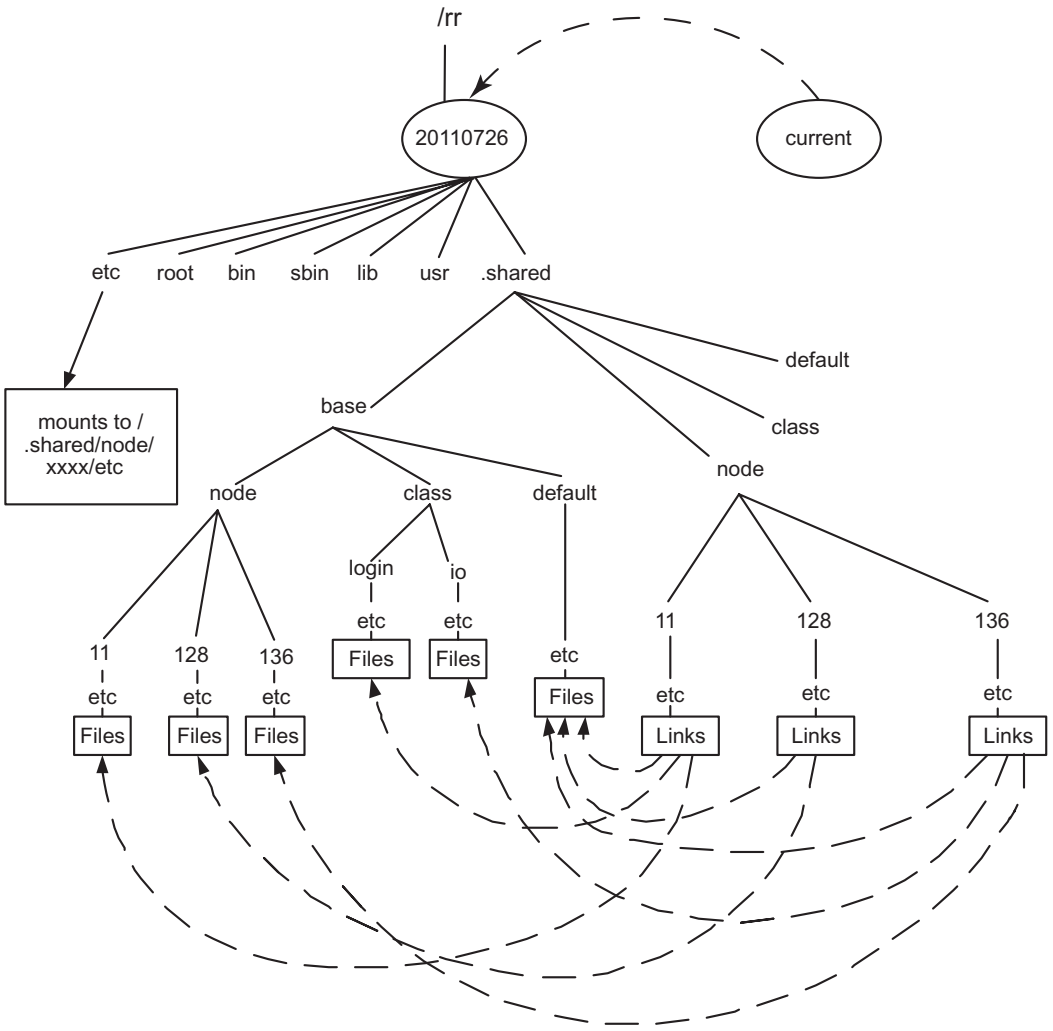
Table 3. File Specialization by Node

File or Directory	Reason for Specialization
`/etc/cron*`	Certain service nodes, such as `sdb` and `syslog`, need custom crontabs.
`/etc/ntp.conf`	A node that runs an NTP server needs a different configuration than NTP clients.
`/etc/sysconfig/network/*`	Each network node should have a different IP address.
`/etc/syslog-ng/syslog-ng.conf.in`	A node that runs a syslog server needs a different configuration than syslog clients.
`/etc/ssh/*key*`	Use when sharing keys across systems is unacceptable.

6.1.2 Visible Shared-root File System Layout

Figure 3 is a detailed illustration of shared-root directory structure. The directory `current` is a subdirectory of `/rr`. The `current` directory links to a time-stamped directory (in this example 20110726). The timestamp indicates the date of the software installation, not the date of the release.

Figure 3. Shared-root Implementation



Service nodes mount the `/rr/current` directory from the boot node as read-only for use as their root file system. The visible file layout, that is, how it appears from the node you are viewing it from, contains the following files:

`/`	Root file system
`/root`	Equivalent to directory of the same name in `/rr/current`
`/bin`	Equivalent to directory of the same name in `/rr/current`
`/lib`	Equivalent to directory of the same name in `/rr/current`
`/sbin`	Equivalent to directory of the same name in `/rr/current`
`/usr`	Equivalent to directory of the same name in `/rr/current`
`/opt`	Equivalent to directory of the same name in `/rr/current`

`/etc`	Contains links to the shared-root files
`/home`	Link to `/home`, a customer-specific location
`/tmp`	Implemented through the `tmpfs` (in RAM)
`/var`	Directory in the `tmpfs` and RAMFS but populated with skeleton files if you do not have persistent `/var`
`/proc`	Per-node pseudo-file system
`/dev`	Per-node pseudo-file system implemented through the DEVFS
`/ufs`	Mount point for the `/ufs` file system to be mounted from the `ufs` node

6.1.3 How Specialization Is Implemented

The shared-root file system is implemented in the `/.shared` directory. Only the `/etc` directory has been set up for specialization. Files in `/etc` are symbolic links to files in `/.shared/base`. A specialized file is a unique version of the file in the `/.shared/base` directory.

The `/.shared` directory contains four subdirectories: `base`, `node`, `class`, and `default`. The `node`, `class`, and `default` directories are also known as `view` directories, because you can look at the file system (with the `xtopview` command) as if the view directory were `/`.

The `base` subdirectory also contains subdirectories called `node`, `class`, and `default`. These are referred to as `base` directories. They contain files that are specific to a certain node, specific to a class of nodes, or shared as the default among all nodes. Under each of the `base` directories is a rooted directory hierarchy where files are stored.

Example 64. Shared-root links

The path of the link shows the type of specialization for the file.

Default specialization:

```
default/: # ls -la /etc/hosts
lrwxrwxrwx 1 root root 31 Dec 8 17:12 /etc/hosts -> /.shared/base/default/etc/hosts
```

Class specialization:

```
class/login/: # ls -la /etc/security/access.conf
lrwxrwxrwx 1 root root 46 Dec 8 17:14 /etc/security/access.conf -> \
/.shared/base/class/login/etc/security/access.conf
```

Node specialization:

```
node/128/: # ls -la /etc/resolv.conf
lrwxrwxrwx 1 root root 36 Dec 8 17:15 /etc/resolv.conf -> \
/.shared/base/node/128/etc/resolv.conf
```

6.1.4 Working with the Shared-root File System

CLE commands shown in Table 4 control and monitor the shared-root file system. For more information, refer to the sections noted and the related man pages.

Table 4. Shared-root Commands

Command	Function
`xtopview`	View file layout from the specified node (see Managing System Configuration with the `xtopview` Tool on page 133).
`xtopcommit`	Record file specialization before leaving `xtopview` shell (see Updating Specialized Files From Within the `xtopview` Shell on page 136).
`xtspec`	Specialize; create a directory structure that links files to non-default files (see Specializing Files on page 136).
`xthowspec`	Determine the type of specialization (see Determining Which Files Are Specialized on page 138).
`xtverifyshroot`	Verify that node-specialized and class-specialized files are linked correctly (see Checking Shared-root Configuration on page 140).
`xtverifyconfig`	Verify that start/stop links generated by tools such as `chkconfig` are consistent across all views of the shared root. You can configure `xtopview` to invoke `xtverifyconfig` automatically; this is the preferred usage. `xtverifyconfig` is not intended for direct use. (See Verifying the Coherency of `/etc/init.d` Files Across All Shared Root Views on page 140.)
`xtverifydefaults`	Verify and fix inconsistent system default links within the shared root. You can configure `xtopview` to invoke `xtverifydefaults` automatically; this is the preferred usage. `xtverifydefaults` is not intended for direct use. (See Verifying the Coherency of `/etc/init.d` Files Across All Shared Root Views on page 140.)
`xtcloneshared`	Create a directory structure for a new node or class based on an existing node or class (see Cloning a Shared-root Hierarchy on page 141).
`xtnce`	Modify the class of a node or display the current class of a node (see Changing the Class of a Node on page 141).

Command	Function
`xtunspec`	Remove specialization (see Removing Specialization on page 142).
`xtoprlog`	Display RCS log information for shared root files (see Displaying RCS Log Information for Shared Root Files on page 142).
`xtopco`	Check out (restore) RCS versioned shared root files (see Checking Out an RCS Version of Shared Root Files on page 143).
`xtoprdump`	Print a list of file specifications that can be used as the list of files to operate on an archive of shared root file system files (see Listing Shared Root File Specification and Version Information on page 144).
`xtoparchive`	Perform operations on an archive of shared root configuration files (see Performing Archive Operations on Shared Root Files on page 145).

6.1.4.1 Managing System Configuration with the `xtopview` Tool

The `xtopview` tool manages the files in the shared-root file system. You specify the view of the system you want, such as from a particular node, when you invoke the command. The system appears as if you were logged in directly to the location you specify; that is, the files that are specialized for that node appear in the `/etc` directory. You can specify location by node ID or hostname.

Changes you make within `xtopview` are logged to a revision control system (RCS) file. When you exit the shell, you are prompted to type a message about each change you have made. Use the `c` command to comment the work you have done in `xtopview`. This information is saved in the Revision Control System (RCS) files.

Tip: Use the `-m msg` option when starting an `xtopview` session to make similar changes to multiple files.

The changed files and messages are then logged to create a history that is stored in the `/.shared/base` directory by its specialization (node, class, or default) and file name. For example, changes and messages relating to default-specialized file `/etc/spk` are stored in `/.shared/base/default/etc/RCS`. Use standard RCS tools, such as `rlog`, for retrieving information.

Warning: If you do not want the changes you have made in your `xtopview` session, you must invoke any necessary commands to undo them. There is no automatic way to back out.

Cray recommends that you configure the shared root from within the `xtopview` shell. Only operations that take place within the `xtopview` shell are logged. If you choose to use specialization commands outside of `xtopview`, they are not logged. Logs reside in the `/rr/current/.shared/log` path relative to the boot node.

New files that are created from within the `xtopview` shell automatically have the specialization that is associated with the view under which you are operating. You do not have to specialize them. If you want a file to be used by all service nodes, create the file in the default view.

The `xtopview` command is typically executed on the boot node; however, you may perform `xtopview` work from the SMW if the system is not booted, for example, if your system is undergoing hardware maintenance. For more information, see Example 70.

Example 65. Starting the `xtopview` shell for a node

To start the `xtopview` shell for node 131, type:

```
boot:~ # xtopview -n 131
node/131/: #
```

Example 66. Starting the `xtopview` shell for a class of nodes

To start the `xtopview` shell for the login nodes, type:

```
boot:~ # xtopview -c login
class/login/: #
```

Note: If you are using the `emacs` editor within the `xtopview` shell, you may see the following message:

```
Symbolic link to RCS-controlled source file; follow link [yes or no]?
```

The symbolic link points to a real file in the `/.shared` directory. If you choose `yes`, you edit the file directly. If you choose `no`, you replace the symbolic link with a real file, but when you exit the `xtopview` shell, the file is moved to the correct location and the link is recreated. The difference is that if you are editing the real file, modifications appear immediately in other views.

Example 67. Starting the `xtopview` shell for a directory other than `/rr/current`

To start the `xtopview` shell in a directory other than `/rr/current`, which is a link to the most current directory, type:

```
boot:~ # xtopview -r /rr/20120901
default/:/ #
```

Example 68. Sample `xtopview` session

```
boot:~ # xtopview -n 3
node/3:/ # vi etc/fstab
. . . (edited the file)
node/3:/ # exit
exit
***File /etc/fstab was MODIFIED
operation on file /etc/fstab? (h for help):c
enter description, terminated with single '.' or end of file:
>changed the fstab file to add support for xyz.
boot:~ #
```

Generally, the `xtopview` command obtains node and class information from the SDB. If the SDB is not running, you can direct `xtopview` to access the `/etc/opt/cray/sdb/node_classes` file by selecting the `-x` option.

Example 69. Starting `xtopview` using `node_classes` for information

For nodes:

```
boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -n 4
```

For classes:

```
boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -c login
```

Example 70. Running `xtopview` from the SMW while the system is not booted

You may use `xtopview` from the SMW to perform software work while the system is not booted. This can be quite useful during hardware maintenance periods.

As `root` on the SMW, verify that the boot node is down:

```
smw:~ # ping boot
```

Mount the boot root and shared root file systems (if they are not already mounted). This example uses the default `bootroot_dir` and the shared root mount point is `/rr`.

```
smw:~ # mount /dev/disk/by-id/your_system_bootroot_ID /bootroot0
smw:~ # mount /dev/disk/by-id/your_system_shareroot_ID /bootroot0/rr
```

Start a default view `xtopview` session:

Note: Additional options like `-c class` or `-n nid` can be used here as well.

```
smw:~ # chroot /bootroot0
smw:/ # xtopview -x /etc/opt/cray/sdb/node_classes -r /rr/current
```

When you are finished with your changes, exit from the `xtopview` and `chroot` sessions:

```
default/:~ # exit
smw:/ # exit
```

Unmount the boot root and shared root file systems:

```
smw:~ # umount /bootroot0/rr
smw:~ # umount /bootroot0
```

Verify that the boot root and shared root file systems are not mounted:

```
smw:~ # mount
smw:~ # df
```

For more information, see the `xtopview`(8) man page.

6.1.4.2 Updating Specialized Files From Within the `xtopview` Shell

When you exit the `xtopview` shell (see Managing System Configuration with the `xtopview` Tool on page 133), changes you make are propagated to the shared-root file system. Use the `xtopcommit` command to immediately update the shared root with modifications you have made. You do not need to leave the `xtopview` shell.

Example 71. Updating a file within `xtopview` shell

```
boot:~ # xtopview -n 3
node/3:/ # vi /etc/fstab
node/3:/ # xtopcommit
***File /etc/fstab was MODIFIED
operation on file /etc/fstab? (h for help):h
c:check-in - record changes in RCS file
d:diff - diff between file and backup RCS file
h:help - print this help message
m:message - set message for later checkins
M:nomsg - clear previously set message
l:list - list file info (ls -l)
s:skip - check-in file with empty log message
q:quit - check-in ALL files without querying
```

6.1.4.3 Specializing Files

Specifying a view with the `xtopview` command does not automatically specialize existing files. To specialize existing files, you must use the specialization command `xtspec`. The command runs on the boot node and creates a copy of a file that is unique to a node or class. The `xtspec` command has the form:

`xtspec [`*options*`]` *file*

The command specializes the file at the location *file* and updates each node or class of nodes that contains the newly specialized file if the new file is the most specialized file in its view. For example, if a file is specialized by class `io`, for all nodes with class `io` the symbolic links associated with this file are updated to point to the new file unless they are already specialized by node (see Figure 2), which is a more restrictive class.

If you are not within `xtopview` (see Managing System Configuration with the `xtopview` Tool on page 133) when you specialize a file, you must specify the path of the shared root with the `-r` option. In addition, the RCS log of changes has a generic entry for each file.

Note: The `xtspec` command can be used only to specify files or directories residing in or under the `/etc` directory. If you attempt to specify a file or directory outside of the `/etc` directory, the command fails and an error message is generated.

The `-V` option of the `xtspec` command specifies the location from which the file that is to be the specialized file is copied. If the `-V` option is specified, the newly specialized file is a duplicate of the file from the target's view. If the `-V` option is not specified, the newly specialized file is a duplicate of the file from the default view.

If you do not specialize a file, the default specialization level is based on the current view if you are running in the `xtopview` shell (see Managing System Configuration with the `xtopview` Tool on page 133) or on the default view if you are operating outside the `xtopview` shell.

Classes are defined in the `node_classes` file (see Class Name on page 58).

Procedure 21. Specializing a file by class login

1. To specialize the file `/etc/dhcpd.conf` by the class of login nodes, enter the login shell.

   ```
   boot:~ # xtopview -c login
   ```

2. Specialize the selected file.

   ```
   class/login:~ # xtspec /etc/dhcpd.conf
   ```

3. Edit `/etc/dhcpd.conf` if it is the default copy of the file. If you have pointed to a unique copy of the file in the `xtspec` command, omit this step.

As a result of this procedure, each node in the class `login` links to the "new" `/etc/dhcpd.conf` file unless the node is already specialized by node. For example, node 23 might already be specialized and link to a different `/etc/dhcpd.conf` file.

Procedure 22. Specializing a file by node

1. To specialize the file `/etc/motd` for node `11`, enter the login shell.

   ```
   boot:~ # xtopview -n 4
   ```

2. Specialize the selected file.

   ```
   node/11/: # xtspec /etc/motd
   ```

This procedure creates a node-specific copy of `/etc/motd`. That is, the directory entry in the `/etc` file associated with `node 11` is updated to point to the new version of `/etc/motd` and the activity is logged.

Procedure 23. Specializing a file by node without entering `xtopview`

- Specify the root path and view mode.

```
boot:~ # xtspec -r /rr/current -V -n 4 /etc/motd
```

As a result of this procedure, the directory entry in the `/etc` file associated with `node 11` is updated to point to the new version of `/etc/motd` but the activity is not logged.

After you have specialized nodes, you can unspecialize them (see `xtunspec` command, Removing Specialization on page 142) or determine how they are specialized (see `xthowspec` command Determining Which Files Are Specialized on page 138). You can also view or change the class type of a particular node (see `xtnce` command, Changing the Class of a Node on page 141).

You can use specialization commands only from the boot node. You must be `root` user to use them. For more information, see the `shared_root`(5) and `xtspec`(8) man pages.

6.1.4.4 Determining Which Files Are Specialized

The CLE `xthowspec` command displays how the files in a specified path are specialized. For example, you might use this command to examine restrictions on login nodes.

The `xthowspec` command has the form:

`xthowspec [`*options*`]` *path*

You can display file specialization for all nodes or all classes, for a particular node or class, for the default view, or for a selection of parameters. Inside the `xtopview` shell, the `xthowspec` command acts on files in the current view by default.

Output has the form *TYPE*:*ITEM*:*FILE*:*SPEC*, where the fields are as follows:

TYPE	Node, class or default.
ITEM	The specific node or class type; this field is empty for the default view.
FILE	The file upon which the command is acting.
SPEC	The specialization level of the file in the view; for example, for default view this is default; for class view options are class or default.

Procedure 24. Finding files in `/etc` that are specialized by a node

1. Enter the `xtopview` shell for the node.

```
boot:~ # xtopview -n 4
```

2. Use the `xthowspec` command for the node.

```
node/4/: # xthowspec -t node /etc
node:4:/etc/fstab.h:node
node:4:/etc/hostname:node
```

Or, outside the `xtopview` shell:

```
boot:~ # xthowspec -r /rr/current -t node -n 4 /etc
node:4:/etc/fstab.h:node
node:4:/etc/hostname:node
```

Example 72. Finding files in `/etc` that are specialized by class

To find all files specialized by class, type:

```
class/login:~ # xthowspec -r /rr/current -t class /etc
node:4:/etc/init.d/rc3.d/K01pbs:class
node:4:/etc/init.d/rc3.d/S11pbs:class
node:16:/etc/init.d/rc3.d/K01pbs:class
node:16:/etc/init.d/rc3.d/S11pbs:class
class:login:/etc/HOSTNAME:class
class:login:/etc/sysconfig/network/routes:class
...
```

Example 73. Finding specialization of a file on a node

To find the specialization of `/etc/dhcpd.conf` on node 4, type:

```
boot:~ # xtopview -n 4
node/4/: # xthowspec /etc/dhcpd.conf
node:4:/etc/dhcpd.conf:default
```

Example 74. Finding nodes on which a file is specialized

To find the nodes that the `/etc/fstab` is specialized on, type:

```
boot:~ # xthowspec -r /rr/current -N /etc/fstab
node:0:/etc/fstab:default
node:1:/etc/fstab:default
node:8:/etc/fstab:class
node:9:/etc/fstab:node
```

To examine specialization outside the `xtopview` shell, you must type the full path name.

Example 75. Finding specialization of a file on a node without invoking the `xtopview` shell

To find the specialization of `/etc/fstab` on node `102`, type:

```
boot:~ # xthowspec -r /rr/current -n 102 /etc/fstab
node:102:/etc/fstab:node
```

Example 76. Finding specialization of files by class without invoking the `xtopview` shell

To find all files that are specialized by class in `/etc` for all nodes, type:

```
boot:~ # xthowspec -r /rr/current -N -t class /etc
node:11:/etc/crontab:class
node:1:/etc/crontab:class
```

For more information, see the `xthowspec`(8) man page.

6.1.4.5 Checking Shared-root Configuration

You can check the configuration of the shared-root file system with the `xtverifyshroot` command:

```
xtverifyshroot [options] path
```

If there are node-specialized or class-specialized files, the command verifies that they are linked correctly. If a problem is detected with a file, it is reported but not corrected.

Note: You must be in the `xtopview` shell to use the `xtverifyshroot` command.

For more information, see the `xtverifyshroot`(8) man page.

6.1.4.6 Verifying the Coherency of `/etc/init.d` Files Across All Shared Root Views

The `xtopview` command is configured to invoke the `xtverifyconfig` utility automatically to resolve potential inconsistencies in the mechanism used to configure various CLE software services on or off.

Note: This is the preferred usage; the `xtverifyconfig` utility is not intended for direct use.

When you use the `chkconfig` utility to configure services on or off, a collection of encoded symbolic links are generated to determine which system services are started or shut down and in what order. The `chkconfig` utility does not account for the multiple levels of specialization within the shared root when `xtopview` is used. As a result, `chkconfig` occasionally produces a startup or shutdown order that violates dependencies between services when all levels of specialization are taken into account. To resolve this problem, you can configure `xtopview` to invoke the `xtverifyconfig` verification utility upon exit. The `xtverifyconfig` utility will detect inconsistencies and may rename startup and shutdown links to maintain the proper dependency ordering. The `/.shared/log` log file in the shared root contains a log of modifications `xtverifyconfig` makes to the shared root.

The xtopview command will run xtverifyconfig upon exit if the XTOPVIEW_VERIFY_INITD environment variable is non-zero when xtopview is invoked, or if the XTOPVIEW_VERIFY_INITD variable is set to non-zero in the /etc/sysconfig/xt file on the boot node. By default, this parameter is not included in the configuration file and this feature is not enabled.

For more information, see the xtverifyconfig(8) man page.

6.1.4.7 Cloning a Shared-root Hierarchy

You can create a directory structure for a new node or class name in the shared-root hierarchy based on an existing node or class with the xtcloneshared command. For more information, see the xtcloneshared(8) man page.

6.1.4.8 Changing the Class of a Node

If you remove nodes, you may need to change the class of the remaining nodes. If you add a login node, you must add it to class login. The xtnce command displays the current class of a node or modifies its class. The command has the form:

```
xtnce [options] nodename
```

Example 77. Finding the class of a node

To identify the class of node 750, type:

```
boot:~ # xtnce 750
750:snx-lnet
```

Example 78. Adding a node to a class

Enter xtopview and use the xtnce command for the node and specify the class it should be:

```
boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes
default/:/ # xtnce -c login 104
```

You also need to change /etc/opt/cray/sdb/node_classes on the boot node so the data is preserved across a boot; this is because the node_classes file is used to initialize the SDB data on the next boot, and the boot node file cannot be updated from within xtopview.

Note: If you make changes to /etc/opt/cray/sdb/node_classes, you **must** make the same changes to the node class settings in CLEinstall.conf before performing an update or upgrade installation; otherwise, the install utility will complain about the inconsistency.

For more information, see the xtnce(8) man page.

Note: The xtnodeclasses2db command inserts the node-class list into the database, but it does not make any changes to the shared root.

6.1.4.9 Removing Specialization

If you specialized a node or class of nodes and, for example, you want to remove unique start-up scripts from them, you can remove this specialization with the `xtunspec` command:

`xtunspec [`*options*`] `*path*

You can unspecialize files for all nodes and classes (default), for a specified class of nodes or for a particular node. Cray strongly recommends that you unspecialize files from within the `xtopview` shell; if you do not unspecialize your files from within the `xtopview` shell (see Managing System Configuration with the `xtopview` Tool on page 133), you must also specify the path for the shared root.

Note: You can only use `xtunspec` on the boot node.

Example 79. Removing node specialization

To remove all versions of `/etc/fstab` specialized by node, type:

```
boot:~ # xtopview
default/:/ # xtunspec -N /etc/fstab
```

Each node is updated so that it uses a version of `/etc/fstab` based on its class, or if that is not available, based on the default version of `/etc/fstab`.

Example 80. Removing class specialization

To remove all versions of `/etc/fstab` that are specialized by, for example, class I/O (`io`), type:

```
boot:~ # xtopview
default/:/ # xtunspec -c io /etc/fstab
```

I/O nodes that link to the class-specialized version of the file are changed to link to the default version of `/etc/fstab`. However, I/O nodes that already link to node-specialized versions of `/etc/fstab` are unchanged. To remove a file specialized by node, you must use the `xtunspec` command on the node (see Example 79).

For more information, see the `xtunspec`(8) man page.

6.1.4.10 Displaying RCS Log Information for Shared Root Files

The `xtoprlog` command displays Revision Control System (RCS) log information for shared root files. Specify the file name using the required *filename* command-line argument. The `xtoprlog` command can be executed from within an `xtopview` shell or from the boot node as `root`. If `xtoprlog` is executed from within `xtopview`, it will operate on the current view; if the command is executed outside of `xtopview` on the boot node, then you must specify a view to use with the `-d`, `-c`, or `-n` options and also the `xtopview` root location with the `--root` option.

The scope of this tool is limited to identification and manipulation of `/etc` configuration data within the shared root. Configuration files on the boot root file system or on the SMW are not managed by this utility.

For more information, see the `xtoprlog`(8) man page.

Example 81. Printing the latest version of a file

Use the `xtoprlog --version` option to print the latest version (revision) number of a specified file:

```
default/:/ # xtoprlog --version /etc/fstab
1.1
```

Example 82. Printing the RCS log for `/etc/fstab` in the node 3 view

Use the `xtoprlog -n` option to specify the `/etc/fstab` node view RCS log to print:

```
default/:/ # xtoprlog -n 3 /etc/fstab
RCS file: /.shared/base/node/3/etc/RCS/fstab,v
Working file: /.shared/base/node/3/etc/fstab
head: 1.6
...
```

Example 83. Displaying differences between two versions of the `/etc/fstab` file

Use the `xtoprlog -x` option with the `xtoprlog -r` option to display the differences between the current version of `/etc/fstab` and version 1.3:

```
default/:/ # xtoprlog -x -r 1.3 /etc/fstab
===================================================================
RCS file: /.shared/base/default/etc/RCS/fstab,v
retrieving revision 1.3
diff -r1.3 /.shared/base/default/etc/fstab
1,3c1,4
< # Default view fstab file 1.3
---> # Default view fstab file 1.7
```

6.1.4.11 Checking Out an RCS Version of Shared Root Files

Use the `xtopco` command to check out a version of shared root files. The `xtopco` command should be run on the boot node using the `xtopview` utility in the default view.

The scope of this tool is limited to identification and manipulation of `/etc` configuration data within the shared root. Configuration files on the boot root file system or on the SMW are not managed by this utility.

Example 84. Checking out a version 1.2 copy of `/etc/fstab`

Use the `xtopco -r` option to specify the version of the file to check out:

```
boot:~ # xtopview
default/:/ # xtopco -r 1.2 /etc/fstab
```

Example 85. Recreating the file link for `/etc/fstab` to the current view's `/etc/fstab` file

To recreate the file link only, use the `xtopco --link` option:

```
boot:~ # xtopview
default/:/ # xtopco --link /etc/fstab
```

For more information, see the `xtopco`(8) man page.

6.1.4.12 Listing Shared Root File Specification and Version Information

Using RCS information, combined with the `xtopview` specialization information, `xtoprdump` prints a list of file specifications that can be used as the list of files to operate on an archive of shared root file system files. The `xtoprdump` command should be invoked using the `xtopview` utility unless the `--root` option is specified.

The scope of this tool is limited to identification and manipulation of `/etc` configuration data within the shared root. Configuration files on the boot root file system or on the SMW are not managed by this utility.

Example 86. Printing specifications for login class specialized files

Use the `xtoprdump -c` option to specify the class view; set to `login` to print the login class specifications:

```
boot:~ # xtopview
default/:/ # xtoprdump -c login
class:login:/etc/HOSTNAME:1.2:*
class:login:/etc/fstab:1.2:*
class:login:/etc/fstab.old:1.1:*
class:login:/etc/modprobe.conf.local:1.2:*
class:login:/etc/opt/cray/modules/Base-opts.default.local:1.2:*
class:login:/etc/sysconfig/network/config:1.2:*
class:login:/etc/sysconfig/network/routes:1.1:*
class:login:/etc/yp.conf:1.1:*
```

Example 87. Printing specifications for all node specialized files

Use the `xtoprdump -n` option to specify the node view; set to `all` for all nodes:

```
boot:~ # xtopview
default/:/ # xtoprdump -n all
node:22:/etc/sysconfig/network/ifcfg-ib0:1.1:*
node:23:/etc/sysconfig/network/ifcfg-ib0:1.1:*
node:30:/etc/new_file:1.1:*
node:30:/etc/opt/cray/rsipd/rsipd.conf:1.1:*
node:30:/etc/sysconfig/network/ifcfg-eth0:1.1:*
node:30:/etc/sysctl.conf:1.2:*
node:30:/etc/sysctl.conf.14524:1.1:*
node:30:/etc/udev/rules.d/77-network.rules:1.1:*
node:5:/etc/exports:1.2:*
node:5:/etc/fstab:1.5:*
node:5:/etc/fstab.old:1.4:*
node:5:/etc/init.d/boot.local:1.1:*
node:5:/etc/motd:1.2:*
node:5:/etc/sysconfig/syslog:1.1:*
node:5:/etc/syslog-ng/syslog-ng.conf:1.2:*
node:8:/etc/sysconfig/network/ifcfg-ib0:1.1:*
node:9:/etc/sysconfig/network/ifcfg-ib0:1.1:*
```

Example 88. Printing specifications for files modified in the default view and include any warning messages

The following `xtoprdump` command prints specifications for modified files (`-m` option) in the default view (`-d` option), including warning messages (`-w` option):

```
boot:~ # xtopview
default/:/ # xtoprdump -m -d -w
default::/etc/alps.gpus:1.2:*
default::/etc/bash.bashrc.local:1.5:*
default::/etc/bash.bashrc.local.rpmsave:1.2:*
...
```

For more information, see the `xtoprdump`(8) man page.

6.1.4.13 Performing Archive Operations on Shared Root Files

Use the `xtoparchive` command to perform operations on an archive of shared root configuration files. Run the `xtoparchive` command on the boot node using the `xtopview` utility in the default view. The archive is a text-based file similar to a tar file and is specified using the required `archivefile` command-line argument. The `xtoparchive` command is intended for configuration files only. Binary files will not be archived. If a binary file is contained within a specification file list, it will be skipped and a warning will be issued.

The scope of this tool is limited to identification and manipulation of `/etc` configuration data within the shared root. Configuration files on the boot root file system or on the SMW are not managed by this utility.

Example 89. Adding files specified by specifications listed in `specfile` to an archive file

Use the following `xtoparchive` command to add files specified by the specifications listed in `specfile` to the archive file `archive.20110422`; create the archive file if it does not already exist:

```
boot:~ # xtopview
default/:/ # xtoparchive -a -f specfile archive.20110422
```

Example 90. Listing specifications for files currently in the `archive.20110422` archive file

Use the `xtoparchive -l` command to list specifications for files currently in the archive file `archive.20110422`:

```
boot:~ # xtopview
default/:/ # xtoparchive -l archive.20110422
```

For more information, see the `xtoparchive`(8) man page.

6.1.5 Logging Shared-root Activity

All specialization activity is logged in the log file `/.shared/log`, which tracks additions, deletions, and modifications of files. To view the details of your changes, you must access the RCS logs that were created during the `xtopview` session.

Note: If you have exited `xtopview` with `Ctrl-c`, you do not log the operations you performed within the shell, The changes to the system are present nonetheless. This means that if you want to back out of changes, it is not sufficient to exit `xtopview`. You must submit the commands to undo what you have done.

6.2 PBS Professional Licensing Requirements for Cray Systems

The licensing scheme for PBS Professional uses a central license server to allow licenses to float between servers. The PBS server and scheduler are run on the service database (SDB) node, therefore, network connectivity must exist between the license server and the SDB node. For information about network configuration options for PBS, see Appendix F, PBS Professional Licensing for Cray Systems on page 407.

6.3 Disabling Secure Shell (SSH) on Compute Nodes

By default, the SSH daemon, `sshd`, is enabled on compute nodes. To disable `sshd` follow this procedure.

Procedure 25. Disabling SSH daemon (sshd) on CNL compute nodes

1. Edit the CLEinstall.conf file and set ssh_generate_root_sshkeys to no (by default, this is set to yes).

   ```
   smw:~ # vi CLEinstall.conf
   ssh_generate_root_sshkeys=no
   ```

2. Invoke the CLEinstall program on the SMW; you must specify the *xtrelease* that is currently installed on the system set that you are using and located in the CLEmedia directory.

   ```
   smw:~ # /home/crayadm/install.xtrelease/CLEinstall --upgrade \
   --label=system_set_label --XTrelease=xtrelease \
   --configfile=/home/crayadm/install.xtrelease/CLEinstall.conf \
   --CLEmedia=/home/crayadm/install.xtrelease
   ```

3. Type y and press the Enter key to proceed when prompted to update the boot root and shared root.

   ```
   *** Do you wish to continue? (y/n) --> y
   ```

   Upon completion, CLEinstall lists suggested commands to finish the installation. Those commands are also described here. For more information about running the CLEinstall program, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

4. Rebuild the boot image using the /var/opt/cray/install/shell_bootimage_*LABEL*.sh script and the xtbootimg and xtcli commands. Suggested commands are included in output from CLEinstall and shell_bootimage_*LABEL*.sh. For more information about creating boot images, follow Procedure 3 on page 68.

5. Run the shell_post.sh script on the SMW to unmount the boot root and shared root file systems and perform other cleanup as necessary.

   ```
   smw:~# /var/opt/cray/install/shell_post_install.sh
   /bootroot0 /sharedroot0
   ```

6.4 Modifying SSH Keys for Compute Nodes

The dropbear RPM is provided with the CLE release. Using dropbear SSH software, you can supply and generate site-specific SSH keys for compute nodes in place of the keys provided by Cray.

Procedure 26. Using dropbear to generate site-specific SSH keys

Follow these steps to replace the RSA™ and DSA/DSS keys provided by the CLEinstall program.

1. Load the dropbear module.

   ```
   crayadm@smw:~> module load dropbear
   ```

2. Create a directory for the new keys on the SMW.

```
crayadm@smw:~> mkdir dropbear_ssh_keys
crayadm@smw:~> cd dropbear_ssh_keys
```

3. To generate a `dropbear` compatible RSA key, type:

```
crayadm@smw:~/dropbear_ssh_keys> dropbearkey -t rsa -f ssh_host_rsa_key.db
Will output 1024 bit rsa secret key to 'ssh_host_rsa_key.db'
Generating key, this may take a while...
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgwCQ9ohUgsrrBw5GNk7w2H5RcaBGajmUv8XN6fxg/YqrsL4t5
CIkNghI3DQDxoiuC/ZVIJCtdwZLQJe708eiZee/tg5y2g8JIb3stg+ol/9BLPDLMeX24FBhCweUpfGCO6Jfm4
Xg4wjKJIGrcmtDJAYoCRj0h9IrdDXXjpS7eI4M9XYZ
Fingerprint: md5 00:9f:8e:65:43:6d:7c:c3:f9:16:48:7d:d0:dd:40:b7
crayadm@smw:~/dropbear_ssh_keys>
```

To generate a `dropbear` compatible DSS key, type:

```
crayadm@smw:~/dropbear_ssh_keys> dropbearkey -t dss -f ssh_host_dss_key.db
Will output 1024 bit dss secret key to 'ssh_host_dss_key.db'
Generating key, this may take a while...
Public key portion is:
ssh-dss AAAAB3NzaC1kc3MAAACBAMEkThlE9N8iczLpfg0wUtuPtPcpIs7Y4KbG3Wg1T4CAEXDnfMCKSyuCy
21TMAvVGCvYd80zPtL04yc1eUtD5RqEKy0h8jSBs0huEvhaJGHx9FzKfGhWi1ZOVX5vG3R+UCOXG+71wZp3LU
yOcv/U+GWhalTWpUDaRU81MPRLW7rnAAAAFQCEqnqW61bouSORQ52d+MRiwp27MwAAAIEAho69yAfGrNzxEI/
kjyDE5IaxjJpIBF262N9UsxleTX6F65OjNoL84fcKqlSL6NV5XJ5O00SKgTuVZjpXO913q9SEhkcI0Zy0vRQ8
H5x3osZZ+Bq20QWof+CtWTqCoWN2xvne0NtET4lg81qCt/KGRq1tY6WG+a01yrvunzQuafQAAACASXvs8h8AA
EK+3TEDj57rBRV4pz5JqWSlUaZStSQ2wJ3Oy1pIJIhKfqGWytv/nSoWnr8YbQbvH9k1BsyQU8sOc5IJyCFu7+
Exom1yrxq/oirfeSgg6xC2rodcs+jH/K8EKoVtTak3/jHQeZWijRok4xDxwHdZ7e3l2HgYbZLmA5Y=
Fingerprint: md5 cd:a0:0b:41:40:79:f9:4a:dd:f9:9b:71:3f:59:54:8b
crayadm@smw:~/dropbear_ssh_keys>
```

4. As `root`, copy the SSH keys to the boot image template.

   Note: To make these changes for a system partition, rather than for the entire system, replace `/opt/xt-images/templates` with `/opt/xt-images/templates-p`*N*, where *N* is the partition number.

```
crayadm@smw:~/dropbear_ssh_keys> su root
```

For the RSA key:

```
smw:/home/crayadm/dropbear_ssh_keys # cp -p ssh_host_rsa_key.db \
/opt/xt-images/templates/default/etc/ssh/ssh_host_rsa_key
```

For the DSA/DSS key:

```
smw:/home/crayadm/dropbear_ssh_keys # cp -p ssh_host_dss_key.db \
/opt/xt-images/templates/default/etc/ssh/ssh_host_dss_key
```

5. Update the boot image to include these changes; follow the steps in Procedure 2 on page 64.

6.5 Configuring the System Environmental Data Collector (SEDC)

To configure the System Environmental Data Collector (SEDC), which collects data about internal cabinet temperatures, cooling system air pressures, critical voltages, etc., see *Using and Configuring System Environment Data Collections (SEDC)* (S–2491).

6.6 Configuring Optional RPMs in the CNL Boot Image

You can configure which optional RPMs are installed into the CNL boot image for your system in one of two ways. First, several parameters are available in the `CLEinstall.conf` file to control whether specific RPMs are included during installation or upgrade of your system software. When you edit `CLEinstall.conf` prior to running `CLEintall`, set the `CNL_` parameters to either `yes` or `no` to indicate which optional RPMs should be included in your CNL compute node boot images. For example, to include these optional RPMs, change the following lines.

```
CNL_audit=yes
CNL_dvs=yes
CNL_ntpclient=yes
CNL_rsip=yes
```

The second method is to add or remove specific RPMs by editing the `/var/opt/cray/install/shell_bootimage_`*LABEL*`.sh` command used when preparing boot images for CNL compute nodes. Change the settings for these parameters to `y` or `n` to indicate which optional RPMs should be included. For example, to include the optional DVS and RSIP RPMs, change the following lines.

Note: If you make changes to `/var/opt/cray/install/shell_bootimage_`*LABEL*`.sh` directly, it is important that you make similar changes to the `CLEinstall.conf` file in order to avoid unexpected configuration changes during update or upgrade installations.

```
CNL_DVS=y
CNL_RSIP=y
```

6.7 Configuring Memory Control Groups

CLE allows an administrator to force compute node applications to execute within memory control groups. Memory control groups are a Linux kernel feature that can improve the resiliency of the kernel and system services running on compute nodes while also accounting for application memory usage.

Before ALPS launches an application on a compute node, it determines how much memory is available. It then creates a memory control group for the application with a memory limit that is slightly less than the amount of available memory on the compute node. CLE tracks the application's memory usage, and if any allocations meet or try to exceed this limit, the allocation fails and the application aborts.

Since non-application processes execute outside of the memory control group and are not bound to this limit, system services should continue to execute normally during these low memory scenarios, resulting in improved resiliency for the kernel and system services. For details on how the memory control group limit is calculated, see the description of the `-M` option in the `apinit`(8) man page.

Procedure 27. Adjusting the memory control group limit

You adjust the memory control group limit using one of two methods:

1. Edit the `rcad_svcs.conf` in the compute node image in `/opt/xt-images`.

   a. Change the `apinit-M` value in the compute node image `/etc/opt/cray/rca/rcad_svcs.conf` file. The following illustrates the `apinit` line within `rcad_svcs.conf`. The total amount of memory taken by the memory control group is the `-M` value multiplied by the number of cores on the reserved compute node.

   ```
   apinit 0 3 1 0x7000016 0  /usr/sbin/apinit -n -r -M 400k
   ```

   b. Rebuild the compute node and service node boot images as detailed in Preparing a Service Node and Compute Node Boot Image on page 63 to ensure the new value is used whenever the new boot image is used.

   c. Reboot the compute nodes.

   d. To ensure the change is not lost during an upgrade of CLE, copy the modified compute node `/etc/opt/cray/rca/rcad_svcs.conf` file into the default template directory in `/opt/xt-images/templates/`.

2. Alternatively, set the memory control group limit using the `mcgroup` option with the `apmgr` command while the compute node is booted. However, when the compute node is rebooted it will revert to the settings in the compute node image `/etc/opt/cray/rca/rcad_svcs.conf`. See the `apmgr`(8) man page for more details.

Procedure 28. Disabling memory control groups

1. Open the `/etc/opt/cray/rca/rcad_svcs.conf` file in the compute node image and remove or comment-out the `apinit -M` option and corresponding value.

2. Within the compute node image edit `/boot/parameters-cnl` and set the `cgroup_disable` parameter to `memory`:

   ```
   cgroup_disable=memory
   ```

3. Rebuild the compute node boot image as detailed in Preparing a Service Node and Compute Node Boot Image on page 63.

4. Reboot the compute nodes using the newly created boot image.

There is a slightly higher likelihood that some applications will cause the compute nodes to experience OOM (out of memory) conditions if they happen run low on memory and memory control groups are disabled. However, most programs will not see this condition as it is highly dependent on application and site configurations.

6.8 Configuring the Zone Moveable Feature for Compute Nodes

Zone moveable is a Linux kernel feature used to reduce external memory fragmentation. It is not a defragmentation mechanism but can possibly help prevent fragmentation to some degree. The strategy in the zone moveable feature is to separate user from kernel memory. Although this feature may improve performance of applications sensitive to memory fragmentation, it does increase the size of the compute node operating system footprint. Cray therefore leaves zone moveable disabled by default.

Procedure 29. Enabling Zone Moveable

1. Add the following `kernelcore_pct` kernel parameter to the `/opt/xt-images/`*workarea*`/compute/boot/parameters-cnl` file. Cray recommends a compute node kernel memory percentage of 5%.

   ```
   kernelcore_pct=5
   ```

2. Boot the compute nodes with the rebuilt compute node boot image as detailed in Preparing a Service Node and Compute Node Boot Image on page 63.

3. (Optional) You can test if a node has the kernel feature enabled by using `aprun` with the following syntax:

   ```
   % aprun -L nid grep 'zone *Movable' /proc/pagetypeinfo
   ```

To disable the zone movable kernel parameter, remove the `kernelcore_pct` kernel parameter from the `parameters-cnl` mentioned in Procedure 29 on page 151, rebuild the compute node image and reboot the compute nodes.

6.9 Using the cray_pam PAM to Log Failed Login Attempts

The cray_pam module is a Pluggable Authentication Module (PAM) that, when configured, provides information to the user at login time about any failed login attempts since their last successful login. The module provides:

- Date and time of last successful login
- Date and time of last unsuccessful login
- Total number of unsuccessful logins since the user's last successful login

Cray recommends that you configure login failure logging on all service nodes. The RPMs are installed by default on the boot root and shared root file systems.

To use this feature, you must configure the pam_tally and cray_pam PAM modules. The PAM configuration files provided with the CLE software allow you to manipulate a common set of configuration files that will be active for all services.

The cray_pam module requires an entry in the PAM common-auth and common-session files or an entry in the PAM auth section and an entry in the PAM session section of any PAM application configuration file. Use of the common files is typically preferable so that other applications such as su also report failed login information; for example:

```
crayadm@boot:~> su -
2 failed login attempts since last login.
Last failure Thu May  8 11:41:20 2008 from smw.
boot:~ #
```

For each log in attempt, a per-user counter is updated. When a successful log in occurs, the statistics are displayed and the counter is cleared. The default location of the pam_tally counter file is /var/log/faillog. Additionally, cray_pam uses a temporary directory, by default, /var/opt/cray/faillog, to store information about the users. Change these defaults by editing /etc/opt/cray/pam/faillog.conf and by using the file= option for each pam_tally and cray_pam entry. You can find an example faillog.conf file in /opt/cray/pam/*pamrelease-version*/etc.

You can configure a number of nodes to share information by modifying the default location for these directories to use a common set of directories, writable to all nodes. Edit /etc/opt/cray/pam/faillog.conf to reflect an alternate, root-writable directory. Configure pam_tally to save tally information in an alternate location using the file= option; each entry for cray_pam must also include the file= option to specify the alternate location.

Limitations:

- If a login attempt fails, `cray_pam` in the `auth` section creates a temporary file; but because the login attempt failed, the `session` section is not called and, as a result, the temporary file is not removed. This is harmless because the file will be overwritten at the next login attempt and removed at the next successful login.
- Logins that occur outside of the PAM infrastructure will not be noted.
- Host names are truncated after 12 characters. This is a limitation in the underlying `faillog` recording.
- The `cray_pam` module requires `pam_tally` to be configured.

Note: For additional information on using the `cray_pam` PAM module, see the `pam`(8) and `pam_tally`(8) man pages.

Procedure 30. Configuring `cray_pam` to log failed login attempts

1. Edit the `/etc/pam.d/common-auth`, `/etc/pam.d/common-account`, and `/etc/pam.d/common-session` files on the boot node.

   Note: In these examples, the `pam_faillog.so` and `pam.tally.so` entries can include an optional `file=`*/path/to/pam_tally/counter/file* argument to specify an alternate location for the tally file.

   Example 91 shows these files after they have been modified to report failed login using an alternate location for the tally file.

   a. Edit the `/etc/pam.d/common-auth` file and add the following lines as the first and last entries:

```
boot:~ # vi /etc/pam.d/common-auth
auth required pam_faillog.so [file=alternatepath] (as the FIRST entry)
auth required pam_tally.so [file=alternatepath] (as the LAST entry)
```

   Your modified `/etc/pam.d/common-auth` file should look like this:

```
#%PAM-1.0
#
# This file is autogenerated by pam-config. All changes
# will be overwritten.
#
# Authentication-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authentication modules that define
# the central authentication scheme for use on the system
# (e.g., /etc/shadow, LDAP, Kerberos, etc.). The default is to use the
# traditional Unix authentication mechanisms.
#
auth    required        pam_faillog.so
auth    required        pam_env.so
auth    required        pam_unix2.so
auth    required        pam_tally.so
```

b. Edit the /etc/pam.d/common-account file and add the following line as the last entry:

```
boot:~ # vi /etc/pam.d/common-account
account required pam_tally.so [file=alternatepath]
```

Your modified /etc/pam.d/common-account file should look like this:

```
#%PAM-1.0
#
# This file is autogenerated by pam-config. All changes
# will be overwritten.
#
# Account-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authorization modules that define
# the central access policy for use on the system.  The default is to
# only deny service to users whose accounts are expired.
#
account required        pam_unix2.so
account required        pam_tally.so
```

c. Edit the /etc/pam.d/common-session file and add the following line as the last entry:

```
boot:~ # vi /etc/pam.d/common-session
session optional pam_faillog.so [file=alternatepath]
```

Your modified /etc/pam.d/common-session file should look like this:

```
#%PAM-1.0
#
# This file is autogenerated by pam-config. All changes
# will be overwritten.
#
# Session-related modules common to all services

#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define tasks to be performed
# at the start and end of sessions of *any* kind (both interactive and
# non-interactive).  The default is pam_unix2.
#
session required        pam_limits.so
session required        pam_unix2.so
session optional        pam_umask.so
session optional        pam_faillog.so
```

2. Copy the edited files to the shared root by using xtopview in the default view.

```
boot:~ # cp -p /etc/pam.d/common-auth /rr/current/software
boot:~ # cp -p /etc/pam.d/common-account /rr/current/software
boot:~ # cp -p /etc/pam.d/common-session /rr/current/software
boot:~ # xtopview -m "configure login failure logging PAM"
default/:/ # cp -p /software/common-auth /etc/pam.d/common-auth
default/:/ # cp -p /software/common-account /etc/pam.d/common-account
default/:/ # cp -p /software/common-session /etc/pam.d/common-session
```

3. Exit `xtopview`.

```
default/:/ # exit
boot:~ #
```

Example 91. Modified PAM configuration files configured to report failed login by using an alternate path

If you configure `pam_tally` to save tally information in an alternate location by using the `file=` option, each entry for `cray_pam` must also include the `file=` option to specify the alternate location.

Your modified `/etc/pam.d/common-auth` file should look like this:

```
#
# /etc/pam.d/common-auth - authentication settings common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authentication modules that define
# the central authentication scheme for use on the system
# (e.g., /etc/shadow, LDAP, Kerberos, etc.).  The default is to use the
# traditional Unix authentication mechanisms.
#
auth    required        pam_faillog.so file=/ufs/logs/tally.log
auth    required        pam_env.so
auth    required        pam_unix2.so
auth    required        pam_tally.so file=/ufs/logs/tally.log
```

Your modified `/etc/pam.d/common-account` file should look like this:

```
#
# /etc/pam.d/common-account - authorization settings common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authorization modules that define
# the central access policy for use on the system.  The default is to
# only deny service to users whose accounts are expired.
#
account required        pam_unix2.so
account required        pam_tally.so file=/ufs/logs/tally.log
```

Your modified `/etc/pam.d/common-session` file should look like this:

```
#
# /etc/pam.d/common-session - session-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define tasks to be performed
# at the start and end of sessions of *any* kind (both interactive and
# non-interactive).  The default is pam_unix2.
#
session required        pam_limits.so
session required        pam_unix2.so
session optional        pam_umask.so
session optional        pam_faillog.so file=/ufs/logs/tally.log
```

6.10 Configuring cron Services

Optional: Configuring cron services is optional on CLE systems.

The cron daemon is disabled, by default, on the shared root file system and the boot root. It is enabled, by default, on the SMW. Use standard Linux procedures to enable cron on the boot root, following Procedure 31 on page 156.

On the shared root, how you configure cron for CLE depends on whether you have set up persistent /var. If you have persistent /var follow Procedure 32 on page 156; if you have not set up persistent /var, follow Procedure 33 on page 157.

The /etc/cron.* directories include a large number of cron scripts. During new system installations and any updates or upgrades, the CLEinstall program disables execute permissions on these scripts and you must manually enable any scripts you want to use.

Procedure 31. Configuring cron for the SMW and the boot node

Note: By default, the cron daemon on the SMW is enabled and this procedure is required only on the boot node.

1. Log on to the target node as root and determine the current configuration status for cron.

   On the on the SMW:

   ```
   smw:~# chkconfig cron
   cron on
   ```

   On the boot node:

   ```
   boot:~ # chkconfig cron
   cron off
   ```

2. Use the chkconfig command to configure the cron daemon to start. For example, to enable cron on the boot node, type the following command:

   ```
   boot:~ # chkconfig --force cron on
   ```

The cron scripts shipped with the Cray customized version of SLES are located under /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly. The system administrator can enable these scripts by using the chkconfig command. However, if you do not have a persistent /var, Cray recommends that you follow Procedure 33.

Procedure 32. Configuring cron for the shared root with persistent /var

Use this procedure for service nodes by using the shared root on systems that are set up with a persistent /var file system.

1. Invoke the `chkconfig` command in the default view to enable the `cron` daemon.

```
boot:~ # xtopview -m "configuring cron"
default/:/ # chkconfig --force cron on
```

2. Examine the `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, and `/etc/cron.monthly` directories and change the file access permissions to enable or disable distributed cron scripts to meet your needs. To enable a script, invoke `chmod ug+x` to make the script executable. By default, `CLEinstall` removes the execute permission bit to disable all distributed cron scripts.

Caution: Some distributed scripts impact performance negatively on a CLE system. To ensure that all scripts are disabled, type the following:

```
default/:/ # find /etc/cron.hourly /etc/cron.daily \
/etc/cron.weekly /etc/cron.monthly \
-type f -follow -exec chmod ugo-x {} \;
```

3. Exit `xtopview`.

```
default/:/ # exit
boot:~ #
```

Procedure 33. Configuring `cron` for the shared root without persistent `/var`

Because CLE has a shared root, the standard `cron` initialization script `/etc/init.d/cron` activates the `cron` daemon on all service nodes. Therefore, the `cron` daemon is disabled by default and you must turn it on with the `xtservconfig` command to specify which nodes you want the daemon to run on.

1. Edit the `/etc/group` file in the default view to add users who do not have root permission to the "trusted" group. The operating system requires that all `cron` users who do not have root permission be in the "trusted" group.

```
boot:~ # xtopview
default/:/ # vi /etc/group
default/:/ # exit
```

2. Create a `/var/spool/cron` directory in the `/ufs` file system on the `ufs` node which is shared among all the nodes of class `login`.

```
boot:~ # ssh root@ufs
ufs:~# mkdir /ufs/cron
ufs:~# cp -a /var/spool/cron /ufs
ufs:~# exit
```

3. Designate a single login node on which to run the scripts in this directory. Configure this node to start `cron` with the `xtservconfig` command rather than the `/etc/init.d/cron` script. This enables users, including root, to submit `cron` jobs from any node of class `login`. These jobs are executed only on the specified login node.

a. Create or edit the following entry in the `/etc/sysconfig/xt` file in the shared root file system in the default view.

```
boot:~ # xtopview
default/:/ # vi /etc/sysconfig/xt
CRON_SPOOL_BASE_DIR=/ufs/cron
default/:/ # exit
```

b. Start an `xtopview` shell to access all login nodes by class and configure the spool directory to be shared among all nodes of class `login`.

```
boot:~ # xtopview -c login
class/login:/ #
```

c. Edit the `/etc/init.d/boot.xt-local` file to add the following lines.

```
class/login:/ # vi /etc/init.d/boot.xt-local
MYCLASS_NID=`rca-helper -i`
MYCLASS=`xtnce $MYCLASS_NID | awk -F: '{ print $2 }'  | tr -d [:space:]`
CRONSPOOL=`xtgetconfig CRON_SPOOL_BASE_DIR`
if [ "$MYCLASS" = "login" -a -n "$CRONSPOOL" ];then
  mv /var/spool/cron /var/spool/cron.$$
  ln -sf $CRONSPOOL /var/spool/cron
fi
```

d. Examine the `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, and `/etc/cron.monthly` directories and change the file access permissions to enable or disable distributed cron scripts to meet your needs. To enable a script, invoke `chmod ug+x` to make the file executable. By default, `CLEinstall` removes the execute permission bit to disable all distributed cron scripts.

Caution: Some distributed scripts impact performance negatively on a CLE system. To ensure that all scripts are disabled, type the following:

```
class/login:/ # find /etc/cron.hourly /etc/cron.daily \
/etc/cron.weekly /etc/cron.monthly \
-type f -follow -exec chmod ugo-x {} \;
```

e. Exit from the login class view.

```
class/login:/ # exit
boot:~ #
```

f. Use the `xtservconfig` command to enable the `cron` service on a single login node; in this example, node 8.

```
boot:~ # xtopview -n 8
node/8:/ # xtservconfig -n 8 add CRON
node/8:/ # exit
```

The `cron` configuration becomes active on the next reboot. For more information, see the `xtservconfig`(8) man page.

6.11 Configuring the Load Balancer

Optional: The load balancer service is optional on systems that run CLE.

The load balancer can distribute user logins to multiple login nodes, allowing users to connect by using the same Cray host name, for example *xthostname*.

Two main components are required to implement the load balancer, the `lbnamed` service (on the SMW and Cray login nodes) and the site-specific domain name service (DNS).

When an external system tries to resolve *xthostname*, a query is sent to the site-specific DNS. The DNS server recognizes *xthostname* as being part of the Cray domain and shuttles the request to `lbnamed` on the SMW. The `lbnamed` service returns the IP address of the least-loaded login node to the requesting client. The client connects to the Cray system login node by using that IP address.

The CLE software installation process installs `lbnamed` in `/opt/cray-xt-lbnamed` on the SMW and in `/opt/cray/lbcd` on all service nodes. Configure `lbnamed` by using the `lbnamed.conf` and `poller.conf` configuration files on the SMW. For more information about configuring `lbnamed`, see the `lbnamed.conf`(5) man page.

Procedure 34. Configuring `lbnamed` on the SMW

1. (Optional) If site-specific versions of `/etc/opt/cray-xt-lbnamed/lbnamed.conf` and `/etc/opt/cray-xt-lbnamed/poller.conf` do not already exist, copy the provided example files to these locations.

```
smw:~ # cd /etc/opt/cray-xt-lbnamed/
smw:/etc/opt/cray-xt-lbnamed/ # cp -p lbnamed.conf.example lbnamed.conf
smw:/etc/opt/cray-xt-lbnamed/ # cp -p poller.conf.example poller.conf
```

2. Edit the `lbnamed.conf` file on the SMW to define the `lbnamed` host name, domain name, and polling frequency.

   ```
   smw:/etc/opt/cray-xt-lbnamed/ # vi lbnamed.conf
   ```

   For example, if `lbnamed` is running on the host name `smw.mysite.com`, set the login node domain to the same domain specified for the `$hostname`. The Cray system *xthostname* is resolved within the domain specified as `$login_node_domain`.

   ```
   $poller_sleep = 30;
   $hostname = "mysite-lb";
   $lbnamed_domain = "smw.mysite.com";
   $login_node_domain = "mysite.com";
   $hostmaster = "rootmail.mysite.com";
   ```

3. Edit the `poller.conf` file on the SMW to configure the login node names.

```
smw:/etc/opt/cray-xt-lbnamed/ # vi poller.conf
#
# groups
# ------------------------
# login     mycray1-mycray3

mycray1 1 login
mycray2 1 login
mycray3 1 login
```

Note: Because `lbnamed` runs on the SMW, `eth0` on the SMW must be connected to the same network from which users log on to the login nodes. Do not put the SMW on the public network.

Procedure 35. Installing the load balancer on an external "white box" server

Optional: Install `lbnamed` on an external "white box" server as an alternative to installing it on the SMW. **Cray does not test or support this configuration.**

A "white box" server is any workstation or server that supports the `lbnamed` service.

1. Shut down and disable `lbnamed`.

```
smw:~# /etc/init.d/lbnamed stop
smw:~# chkconfig lbnamed off
```

2. Locate the `cray-xt-lbnamed` RPM on the `Cray CLE 5.0.UP`*nn* `Software` media and install this RPM on the "white box." Do **not** install the `lbcd` RPM.

3. Follow the instructions in the `lbnamed.conf`(5) man page to configure `lbnamed`, taking care to substitute the name of the external server wherever `SMW` is indicated, then enable the service.

6.12 Configuring Node Health Checker (NHC)

For an overview of NHC (sometimes referred to as *NodeKARE*), see the `intro_NHC`(8) man page. For additional information about ALPS and how ALPS cooperates with NHC to perform application cleanup, see Chapter 8, Using the Application Level Placement Scheduler (ALPS) on page 261.

6.12.1 `/etc/opt/cray/nodehealth/nodehealth.conf` Configuration File

NHC can be run under two basic circumstances:

- when a node boots
- immediately after applications within a reservation have terminated and immediately after a reservation has terminated

The latter circumstance calls Compute Node Cleanup (CNCU). Its objective is to efficiently return compute nodes to the pool of available nodes with as much free memory as they have when they are first booted. ALPS invokes NHC after every application completes and after every reservation completes. The NHC tests that run after applications are an *application set*. The NHC tests that run after reservations exit are a *reservation set*. With multiple test sets executing, CNCU requires more than one instance of NHC to be running simultaneously. The `advanced_features` NHC control variable must be enabled to use CNCU. The default setting of `advanced_features` in the example NHC configuration file is `on`.

To support running NHC at boot time and after applications and reservations complete, NHC uses two separate and independent configuration files, which enable NHC to be configured differently for these situations.

After application and reservation termination: The configuration file that controls NHC behavior after a job has terminated is `/etc/opt/cray/nodehealth/nodehealth.conf`, which is located in the shared root. The CLE installation and upgrade processes automatically install this file and enable NHC software; there is no need for you to change any installation configuration parameters or issue any commands. However, if you like, you may edit this configuration file to customize NHC behavior. After you do so, the changes you made are reflected in the behavior of NHC the next time that it runs.

When a node boots: The configuration file that controls NHC behavior on boot is located on the compute node. To change this file, you must instead change its template, which is located on the SMW, in one of two locations.

On non-partitioned systems, the SMW template is located here:

```
/opt/xt-images/templates/default/etc/opt/cray/nodehealth/nodehealth.conf
```

On partitioned systems, the SMW template is located here, where p*x*is the partition number:

```
/opt/xt-images/templates/default-px/etc/opt/cray/nodehealth/nodehealth.conf
```

In either case, after you have modified the `nodehealth.conf` file, you must remake the boot image for the compute node and reboot the node with the new boot image in order for your changes to take effect.

Each CLE release package also includes an example NHC configuration file, `/opt/cray/nodehealth/default/etc/nodehealth.conf.example`. The `nodehealth.conf.example` file is a copy of the `/etc/opt/cray/nodehealth/nodehealth.conf` file provided for an initial installation.

Important: The `/etc/opt/cray/nodehealth/nodehealth.conf` file is not overwritten during a CLE upgrade if the file already exists. This preserves your site-specific modifications previously made to the file. However, you should compare your `/etc/opt/cray/nodehealth/nodehealth.conf` file content with the `/opt/cray/nodehealth/default/etc/nodehealth.conf.example` file provided with each release to identify any changes, and then update your `/etc/opt/cray/nodehealth/nodehealth.conf` file accordingly.

If the `/etc/opt/cray/nodehealth/nodehealth.conf` file does **not** exist, then the `/opt/cray/nodehealth/default/etc/nodehealth.conf.example` file is copied to the `/etc/opt/cray/nodehealth/nodehealth.conf` file.

To use an alternate NHC configuration file, use the `xtcleanup_after -f` *alt_NHCconfigurationfile* option to specify which NHC configuration file to use with the `xtcleanup_after` script. For additional information, see the `xtcleanup_after`(8) man page.

NHC can also be configured to automatically dump, reboot, or dump and reboot nodes that have failed tests. This is controlled by the `action` variable specified in the NHC configuration file that is used with each NHC test and the `/etc/opt/cray-xt-dumpd/dumpd.conf` configuration file. For additional information, see Using `dumpd` to Automatically Dump and Reboot Nodes on page 90, the `dumpd`(8) man page, and the `dumpd.conf` configuration file on the System Management Workstation (SMW).

6.12.2 Configuring Node Health Checker Tests

Edit the `/etc/opt/cray/nodehealth/nodehealth.conf` file to configure the NHC tests that will test CNL compute node functionality. All tests that are enabled will run when NHC is in either Normal Mode or in Suspect Mode. Tests run in parallel, independently of each other, except for the `Free Memory Check` test, which requires that the `Application Exited Check` test passes before the `Free Memory Check` test begins.

The `xtcheckhealth` binary runs the NHC tests; for information about the `xtcheckhealth` binary, see the `intro_NHC`(8) and `xtcheckhealth`(8) man pages.

The NHC tests are listed below. In the default NHC configuration file, each test that is enabled starts with an action of `admindown`, except for the `Free Memory Check`, which starts with an action of `log`.

Important: Also read important test usage information in Guidance About NHC Tests on page 166.

- `Accelerator`, which tests the health of any accelerators present on the node. It is an application set test and should not be run in the reservation set.

  The global accelerator test (`gat`) script detects the type of accelerator(s) present on the node and then launches a test specific to the accelerator type. The test fails if it is unable to run successfully on the accelerator, or if the amount of allocated memory on the accelerator exceeds the amount specified using the `gat -m` argument.

  The `Accelerator` test is enabled in the default NHC configuration file.

- `Application Exited Check`, which verifies that any remaining processes from the most recent application have terminated. It is an application set test and should not be run in the reservation set because an application is not associated with a reservation cancellation.

  The `Application Exited Check` test checks locally on the compute node to see if there are processes running under the ID of the application (APID). If there are processes running, then NHC waits a period of time (set in the configuration file) to determine if the application processes exit properly. If the process does not exit within that time, then this test fails.

  The `Application Exited Check` test is enabled in the default NHC configuration file.

- `Apinit Log and Core File Recovery`, which is a plugin script to copy `apinit` core dump and log files to a login/service node. It is an application set test.

  This test is not enabled in the default NHC configuration file. It should not be enabled until after a destination directory is decided on and specified in the NHC configuration file.

- `Apinit Ping`, which verifies that the ALPS daemon is running on the compute node and is responsive. It is an application set test.

  The `Apinit Ping` test queries the status of the `apinit` daemon locally on each compute node; if the `apinit` daemon does not respond to the query, then this test fails.

  The `Apinit Ping` test is enabled in the default NHC configuration file.

- `Free Memory Check`, which examines how much memory is consumed on a compute node while applications are not running. Use it only as a reservation test because an application within a reservation may leave data for the another application in a reservation. If run in the application set, `Free Memory Check` could consider data that was intentionally left for the next application to be leaked memory and mark the node `admindown`. Run the `Free Memory Test` only after the `Reservation` test passes successfully.

The `Free Memory Check` test is enabled in the default NHC configuration file; however, its action is `log` only.

- `Filesystem`, which ensures that the compute node is able to perform simple I/O to the specified file system. It is configured as an application set test in the default configuration, but it can be run in the reservation set. For a file system that is mounted read-write, the test performs a series of operations on the file system to verify the I/O. A file is created, written, flushed, synced, and deleted. If a mount point is not explicitly specified, the mount point(s) from the compute node `/etc/fstabs` file will be used and a `Filesystem` test will be created for each mount point found in the file. If a mount point is explicitly specified, then only that file system will be checked. You can specify multiple `FileSystem` tests by placing multiple `Filesystem` lines in the configuration file. One line could specify the implicit `Filesystem` test. The next line could specify a specific file system that does not appear in `/etc/fstab`. This could continue for any and all file systems.

  If you enable the `Filesystem` test, you can place an optional line (such as, Excluding: `FileSystem-foo`) in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file that allows you to list mount points that should **not** be tested by the `Filesystem` test. This allows you to intentionally exclude specific mount points even though they appear in the `fstab` file. This action prevents NHC from setting nodes to `admindown` because of errors on relatively benign file systems. Explicitly specified mount points cannot be excluded in this fashion; if they should not be checked, then they should simply not be specified.

  The `Filesystem` test creates its temporary files in a subdirectory (`.nodehealth.fstest`) of the file system root. An error message is written to the console when the `unlink` of a file created by this test fails.

  The `Filesystem` test is enabled in the default NHC configuration file.

- `Hugepages`, which calculates the amount of memory available in a specified page size with respect to a percentage of `/proc/boot_freemen`. It is a reservation set test.

  This test will continue to check until either the memory clears up or the time-out is reached. The default time-out is 300 seconds.

  The `Hugepages` test is disabled in the default NHC configuration file.

- `Sigcont Plugin`, which sends a SIGCONT signal to the processes of the current APID. It is an application set test.

  The `Sigcont Plugin` test is disabled in the default NHC configuration file.

- `Plugin`, which allows scripts and executables not built into NHC to be run,

provided they are accessible on the compute node. No plugins are configured by default and the `Plugin` test is disabled in the default NHC configuration file so that local configuration settings may be used.

For information about writing a plugin test, see *Writing a Node Health Checker (NHC) Plugin Test* (S–0023).

- `ugni_nhc_plugins`, which tests the User level Gemini Network Interface (uGNI) on compute nodes. It is a reservation set test and an application set test. By extension, testing the uGNI interface also tests the proper operation of parts of the network interface card (NIC). The test sends a datagram packet out to the node's NIC and back again.
- `Reservation`, which checks for the existence of the `/proc/reservations/`*rid* directory, where *rid* is the reservation ID. It is a reservation set test, and should not be run in the application set.

  If this directory still exists, the test will attempt to end the reservation and then wait for the specified *timeout* value for the directory to disappear. If the test fails and Suspect Mode is enabled, NHC enters Suspect Mode. In Suspect Mode, `Reservation` continues running, repeatedly requesting that the kernel clean up the reservation, until the test passes or until Suspect Mode times out. If the directory does not disappear in that time, the test prints information to the console and exits with a failure.

  The `Reservation` test is enabled in the default NHC configuration file, with a *timeout* value of 300 seconds.
- `CCM plugin`, which validates the cleanup of a cluster compatibility mode (CCM) environment at the end of a reservation. It is a reservation set test, and it will not run if it is misconfigured as an application test.

  This test runs on a compute node only when `/var/crayccm` is detected. The test removes the `/var/lib/{empty,debus}` directories, unmounts CCM mount points if they still exist, and unmounts `/dsl/dev/random` and `/dsl/dev/pts`. If the unmounts are successful, the test removes the `/var/crayccm`, `/var/lib/rpcbind`, and `/var/spool/{PBS,torque}` directories.

  The `CCM plugin` is not included in a site's NHC configuration file. Administrators must add the test to their configuration in order to use it. See the `/opt/cray/nodehealth/default/etc/nodehealth.conf.example` file for `CCM plugin` settings to copy into a site's NHC configuration file.

Individual tests may appear multiple times in the `/etc/opt/cray/nodehealth/nodehealth.conf` file, with different variable values. Every time a test is specified in the `/etc/opt/cray/nodehealth/nodehealth.conf` file, NHC will run that test. This means if the same line is specified five times, NHC will try to run that same test five times. This functionality is mainly used in the case of the `Plugin` test, allowing you to specify as many additional tests as you want to write, or the `Filesystem` test, allowing you to specify as many additional file systems as you want. However, any test can be specified to run any number of times. Different parameters and test actions can be set for each test. For example, this could be used so that you can set up hard limits and soft limits for the `Free Memory Check` test. Two `Free Memory Check` tests could be specified in the configuration file; the first test configured to only warn about small amounts of non-free memory, and the second test configured to `admindown` a node that has large amounts of non-free memory. See the `/etc/opt/cray/nodehealth/nodehealth.conf` file for configuration information.

6.12.2.1 Guidance About NHC Tests

Guidance about the `Accelerator` test: This test uses the global accelerator test (`gat`) script (`/opt/cray/nodehealth/default/bin/gat.sh`) to first detect the accelerator type and then launch the test specific to that type of accelerator. The `gat` script supports two arguments for NVIDIA GPUs:

`-m` *maximum_memory_size*

You can specify the *maximum_memory_size* as either a kilobyte value or a percentage of total memory. For example, `-m 100` specifies that no more than 100 kilobytes of memory can be allocated, while `-m 10%` specifies that no more than 10 percent of memory can be allocated.

In the default NHC configuration file, the specified memory size is `10%`.

`-r` Perform a soft restart on the GPU and then rerun the test. In the default NHC configuration file, the `-r` argument is specified.

The `gat` script has the following options for Intel Xeon Phi:

`-M` *kilobytes* or `-M` *n%*

This option works exactly as the `-m` option for the NVIDIA GPUs.

`-c` Specifies the minimum number of cores that must be active on the Xeon Phi for the test to pass. If `-c` is omitted, the minimum number of active cores required to pass the test is the total number of cores on the Xeon Phi.

Guidance about the `Application Exited Check` and `Apinit Ping` tests: These two tests must be enabled and both tests must have their action set as `admindown` or `die`; otherwise, NHC runs the risk of allowing ALPS to enter a live-lock. (Specify the `die` action only when the `advanced_features` control variable is turned off.) ALPS must guarantee the following two things about the nodes in a reservation before releasing that reservation: 1) ALPS must guarantee that ALPS is functioning on the nodes, and 2) ALPS must guarantee that the previous application has exited from the nodes. Either those two things are guaranteed or the nodes must be set to some state other than `up`. When either ALPS has guaranteed the two things about the nodes or the nodes have been set to some state other than `up`, then ALPS can release the reservation. These two NHC tests guarantee those two things: 1) the `Apinit_ping` test guarantees that ALPS is functioning on the nodes, and 2) the `Application_Exited_Check` test guarantees that the previous application has exited from the nodes. If either test fails, then NHC sets the nodes to `suspect` state (if Suspect Mode is enabled; otherwise, NHC sets the nodes to `admindown` or `unavail`). In the end, either the nodes pass those tests, or the nodes are no longer in the `up` state. In either case, ALPS is free to release the reservation and the live-lock is avoided. However, this only happens if the two tests are enabled and their action is set as `admindown` or `die`. The `log` action does not suffice because it does not change the state of the nodes. If either test is disabled or has an action of `log`, then ALPS may live-lock. In this live-lock, ALPS will call NHC endlessly.

Guidance about the `Filesystem` test: The NHC `Filesystem` test can take an explicit argument (the mount point of the file system) or no argument. If an argument is provided, then the `Filesystem` test is referred to as an explicit `Filesystem` test. If no argument is given, the `Filesystem` test is referred to as an implicit `Filesystem` test.

The explicit `Filesystem` test will test the file system located at the specified mount point.

The implicit `Filesystem` test will test each file system listed in the `/etc/fstab` file on each compute node. The implicit `Filesystem` test is enabled by default in the NHC configuration file.

The `Filesystem` test will determine whether a file system is mounted read-only or read-write. If the file system is mounted read-write, then NHC will attempt to write to it. If it is mounted read-only, then NHC will attempt to read the directory entities "." and ".." in the file system to guarantee, at a minimum, that the file system is readable.

Some file systems are mounted on the compute nodes as read-write file systems, while their underlying permissions are read-only. As an example, for an auto-mounted file system, the base mount-point may have read-only permissions; however, it could be mounted as read-write. It would be mounted as read-write, so that the auto-mounted sub-mount-points could be mounted as read-write. The read-only permissions prevent tampering with the base mount-point. In a case such as this, the `Filesystem` test would see that the base mount-point had been mounted as a read-write file system. The `Filesystem` test would try to write to this file system, but the write would fail due to the read-only permissions. Because the write fails, then the `Filesystem` test would fail, and NHC would incorrectly decide that the compute node is unhealthy because it could not write to this file system. For this reason, file systems that are mounted on compute nodes as read-write file systems, but are in reality read-only file systems, should be excluded from the implicit `Filesystem` test.

You can exclude tests by adding an "Excluding: *file system mount point*" line in the NHC configuration file. See the NHC configuration file for further details and an example.

A file system is deemed a critical file system if it is needed to run applications. All systems will likely need at least one shared file system for reading and writing input and output data. Such a file system would be a critical file system. File systems that are not needed to run applications or read and write data would be deemed as noncritical file systems. You need to determine the criticality of each file system.

Cray recommends the following:

- Excluding noncritical file systems from the implicit `Filesystem` test. See the NHC configuration file for further details and an example.
- If there are critical file systems that do not appear in the `/etc/fstab` file on the compute nodes (such file systems would not be tested by the implicit `Filesystem` test), these critical file systems should be checked through explicit `Filesystem` tests. You can add explicit `Filesystem` tests to the NHC configuration file by providing the mount point of the file system as the final argument to the `Filesystem` test. See the NHC configuration file for further details and an example.
- If you have a file system that is mounted as read-write but it has read-only permissions, you should exclude it from the implicit `Filesystem` test. NHC does not support such file systems.

Guidance about the `Hugepages` test: The `Hugepages` test runs the `hugepages_check` command, which supports two arguments.

`-t` *threshold*

Use this argument to specify the *threshold* as a percentage of `/proc/boot_freemem`. If this test is enabled and this argument is not supplied, the default of `-t 90` is used.

`-s` *size* Specify the hugepage size. The valid sizes are `2`, `4`, `8`, `16`, `32`, `64`, `128`, `256`, and `512`. If this test is enabled and this argument is not supplied, the default of `-s 2` is used.

Guidance about the NHC `Lustre` file system test: The Lustre file system has its own hard time-out value that determines the maximum time that a Lustre recovery will last. This time-out value is called `RECOVERY_TIME_HARD`, and it is located in the file system's `fs_defs` file. The default value for the `RECOVERY_TIME_HARD` is fifteen minutes.

Important: The time-out value for the NHC `Lustre` file system test should be **twice** the `RECOVERY_TIME_HARD` value.

The default in the NHC configuration file is thirty minutes, which is twice the default `RECOVERY_TIME_HARD`. If you change the value of `RECOVERY_TIME_HARD`, you must also correspondingly change the time-out value of the NHC `Lustre` file system test.

The NHC time-out value is specified on this line in the NHC configuration file:

```
# Lustre: <warning time-out> <test time-out> <restart delay>
Lustre: 900 1800 60
```

If you change the `RECOVERY_TIME_HARD` value, you must change the `1800` seconds (thirty minutes) to reflect your new `RECOVERY_TIME_HARD` multiplied by two.

Further, the overall time-out value of NHC's Suspect Mode is based on the maximum time-out value for all of the NHC tests. Invariably, the NHC `Lustre` file system test has the longest time-out value of all the NHC tests.

Important: If you change the NHC `Lustre` file system test time-out value, then you must also change the time-out value for Suspect Mode. The time-out value for Suspect Mode is set by the `suspectend` variable in the NHC configuration file. The guidance for setting the value of `suspectend` is that it should be the maximum time-out value, plus an additional buffer. In the default case, `suspectend` was set to thirty-five minutes – thirty minutes for the Lustre test, plus an additional five-minute buffer. For more information about the `suspectend` variable, see Suspect Mode on page 175.

6.12.2.2 NHC Control Variables

The following variables in `/etc/opt/cray/nodehealth/nodehealth.conf` affect the fundamental behavior of NHC.

`nhcon: [on|off]`

Turning off `nhcon` disables NHC entirely.

Default: `on`

`advanced_features: [on|off]`

If set to `on`, this variable allows multiple instances of NHC to run simultaneously. This variable must be `on` to use CNCU and reservation sets.

Default: `on`

`dumpdon: [on|off]`

If set to `off`, NHC will not request any dumps or reboots from `dumpd`. This is a quick way to turn off dump and reboot requests from NHC. The `dump`, `reboot`, and `dumpreboot` actions do not function properly when this variable is `off`.

Default: `on`

`anyapid: [on|off]`

Turning `anyapid` on specifies that NHC should look for any `apid` in `/dev/cpuset` while running the `Application Exited Check` and print stack traces for processes that are found.

Default:`off`

6.12.2.3 Global Configuration Variables That Affect All NHC Tests

The following global configuration variables may be set in the `/etc/opt/cray/nodehealth/nodehealth.conf` file to alter the behavior of all NHC tests. The global configuration variables are case-insensitive.

`Runtests:` *Frequency*

Determines how frequently NHC tests are run on the compute nodes. *Frequency* may be either `errors` or `always`. When the value `errors` is specified, the NHC tests are run only when an application terminates with a non-zero error code or terminates abnormally. When the value `always` is specified, the NHC tests are run after every application termination. If you do not specify the `Runtests` global variable, the implicit default is `errors`.

This variable applies only to tests in the application set; reservations do not terminate abnormally.

`Connecttime:` *TimeoutSeconds*

Specifies the amount of time, in seconds, that NHC waits for a node to respond to requests for the TCP connection to be established. If Suspect Mode is disabled and a particular node does not respond after `connecttime` has elapsed, then the node is marked `admindown`. If Suspect Mode is enabled and a particular node does not respond after `connecttime` has elapsed, then the node is marked `suspect`. Then, NHC will attempt to contact the node with a frequency established by the `recheckfreq` variable. (For information about Suspect Mode and the `recheckfreq` variable, see Suspect Mode on page 175.)

If you do not specify the `Connecttime` global variable, then the implicit default TCP time-out value is used. NHC will not enforce time-out on the connections if none is specified. The `Connecttime`: *TimeoutSeconds* value provided in the default NHC configuration file is `60` seconds.

The following global variables control the interaction of NHC and `dumpd`, the SMW daemon that initiates automatic dump and reboot of nodes.

`maxdumps:` *MaximumNodes*

Specifies the number of nodes that fail with the `dump` or `dumpreboot` action that will be dumped. For example, if NHC was checking on 10 nodes that all failed tests with the `dump` or `dumpreboot` actions, only the number of nodes specified by `maxdumps` would be dumped, instead of all of them. The default value is `1`.

To disable dumps of failed nodes with `dump` or `dumpreboot` actions, set `maxdumps: 0`.

`downaction:` *action*

Specifies the action NHC takes when it encounters a down node. Valid actions are `log`, `dump`, `reboot`, and `dumpreboot`. The default action is `log`.

`downdumps:` *number_dumps*

Specifies the maximum number of dumps that NHC will dump for a given APID, assuming that the `downaction` variable is either `dump` or `dumpreboot`. These dumps are in addition to any dumps that occur because of NHC test failures. The default value is `1`.

The following global variables control the interaction between NHC, ALPS, and the SDB.

`alps_recheck_max:` *number of seconds*

NHC will attempt to verify its view of the nodes's states with the ALPS view. If NHC is unable to contact ALPS, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

`alps_sync_timeout:` *number of seconds*

If NHC is unable to contact ALPS to verify the nodes's states, this variable controls the length of time before NHC gives up and aborts.

Default value: 1200 seconds

`alps_warn_time:` *number of seconds*

If NHC is unable to contact ALPS to verify the nodes's states, this variable controls how often warnings are issued.

Default value: 120 seconds

`sdb_recheck_max:` *number of seconds*

NHC will contact the SDB to query for the nodes's states. If NHC is unable to contact the SDB, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

`sdb_warn_time:` *number of seconds*

If NHC is unable to contact the SDB, this variable controls how often warnings are issued.

Default value: 120 seconds

`node_no_contact_warn_time:` *number of seconds*

If NHC is unable to contact a specific node, this variable controls how often warnings are issued.

Default value: 600 seconds

The following global variable controls NHC's use of node states.

`unhealthy_state:` *swdown*

When a node is deemed unhealthy, it is normally is set to `admindown`. This variable permits a different state to be chosen instead.

Default: not set

`unhealthy_state:` *rebootq*

When a node is going to be rebooted, it normally is set to `Unavail`. This variable permits a different state to be chosen instead.

Default: not set

6.12.2.4 Standard Variables That Affect Individual NHC Tests

The following variables are used with each NHC test; set each variable for each test. All variables are case-insensitive. Each NHC test has values supplied for these variables in the default NHC configuration file.

Note: Specific NHC tests require additional variables, which are defined in the `nodehealth` configuration file.

action Specifies the action to perform if the compute node fails the given NHC test. *action* may have one of the following values:

- `log` — Logs the failure to the system console log; the `log` action will not cause a compute node's state to be set to `admindown`.

  Important: Tests that have an action of `Log` do **not** run in Suspect Mode. If you use plugin scripts with an action of `Log`, the script will only be run once, in Normal Mode; this makes log collecting and various other maintenance tasks easier to code.

- `admindown` — Sets the compute node's state to `admindown` (no more applications will be scheduled on that node) and logs the failure to the system console log.

  If Suspect Mode is enabled, the node will first be set to `suspect` state, and if the test continues to fail, the node will be set to `admindown` at the end of Suspect Mode.

- `die` — Halts the compute node so that no processes can run on it, sets the compute node's state to `admindown`, and logs the failure to the system console log. (The `die` action is the equivalent of a kernel panic.) This action is good for catching bugs because the state of the processes is preserved and can be dumped at a later time.

  Note: If the `advanced_features` variable is enabled, `die` is not allowed.

Each subsequent action includes the actions that preceded it; for example, the `die` action encompasses the `admindown` and `log` actions.

Note: If NHC is running in Normal Mode and cannot contact a compute node, and if Suspect Mode is not enabled, NHC will set the compute node's state to `admindown`.

The following actions control the NHC and `dumpd` interaction.

- `dump` — Sets the compute node's state to `admindown` and requests a dump from the SMW, in accordance with the `maxdumps` configuration variable.
- `reboot` — Sets the compute node's state to `unavail` and requests a reboot from the SMW. The `unavail` state is used rather than the `admindown` state when nodes are to be rebooted because a node that is set to `admindown` and subsequently rebooted stay in the `admindown` state. The `unavail` state does not have this limitation.
- `dumpreboot` — Sets the compute node's state to `unavail` and requests a dump and reboot from the SMW.

warntime Specifies the amount of test time, in seconds, that should elapse before `xtcheckhealth` logs a warning message to the console file. This allows an administrator to take corrective action, if necessary, before the *timeout* is reached.

timeout Specifies the total time, in seconds, that a test should run before an error is returned by `xtcheckhealth` and the specified *action* is taken.

restartdelay

Valid only when NHC is running in Suspect Mode. Specifies how long NHC will wait, in seconds, to restart the test after the test fails. The minimum restart delay is one second.

sets Indicates when to run a test. The default NHC configuration specifies to run specific tests after application completion and to run an alternate group of tests at reservation end. When ALPS calls NHC at the end of the application, tests marked with `Sets: Application` are run. By default, these tests are: `Filesystem`, `Accelerator`, `ugni_nhc_plugins`, `Application Exited Check`, `Apinit Ping Test`, and `Apinit Log and Core File Recovery`. At the end of the reservation, ALPS calls tests marked `Sets: Reservation`. By default, these are: `Free Memory Check`, `ugni_nhc_plugins`, `Reservation`, and `Hugepages Check`.

If no set is specified for a test, it will default to `Application`, and run when ALPS calls NHC at the end of the application. If NHC is launched manually, using the `xtcheckhealth` command, and the `-m` *sets* argument is not specified on the command line, then `xtcheckhealth` defaults to running the `Application` set.

If a test is marked `Sets: All`, it will always run, regardless of how NHC is invoked.

6.12.3 Suspect Mode

Upon entry into Suspect Mode, NHC immediately allows healthy nodes to be returned to the resource pool. Suspect Mode allows the remaining nodes, which are all in `suspect` state, an opportunity to return to healthiness. If the nodes do not return to healthiness by the end of the Suspect Mode (determined by the `suspectend` global variable; see below), their states are set to `admindown`. For more information about how Suspect Mode functions, see the `intro_NHC`(8) man page.

Important: Suspect Mode is enabled in the default `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file. Cray Inc. recommends that you run NHC with Suspect Mode enabled.

If enabled, the default NHC configuration file provided from Cray Inc. uses the following Suspect Mode variables:

`suspectenable:`

Enables Suspect Mode; valid values are `y` and `n`. The `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file provided from Cray Inc. has this variable set as `suspectenable: y`.

`suspectbegin:`

Sets the Suspect Mode timer. Suspect Mode starts after the number of seconds indicated by `suspectbegin` have expired. The `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file provided from Cray Inc. has this variable set as `suspectbegin: 180`.

`suspectend:`

Suspect Mode ends after the number of seconds indicated by `suspectend` have expired. This timer only starts after NHC has entered Suspect Mode. The `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file provided from Cray Inc. has this variable set as `suspectend: 2100`.

Considerations when evaluating shortening the length of Suspect Mode:

- You can shorten the length of Suspect Mode if you do not have external file systems, such as Lustre, that NHC would be checking.
- The length of Suspect Mode should be at least a few seconds longer than the longest time-out value for any of the NHC tests. For example, if the `Filesystem` test had the longest time-out value at 900 seconds, then the length of Suspect Mode should be at least 905 seconds.
- The longer Suspect Mode is, the longer nodes have to recover from any unhealthy situations. Setting the length of Suspect Mode too short reduces this recovery time and increases the likelihood of the nodes being marked `admindown` prematurely.

`recheckfreq:`

Suspect Mode rechecks the health of the nodes in `suspect` state at a frequency specified by `recheckfreq`. This value is in seconds. The `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file provided from Cray Inc. has this variable set as `recheckfreq: 300`. (For a detailed description about NHC actions during the recheck process, see the `intro_NHC`(8) man page.)

6.12.4 NHC Messages

NHC messages may be found on the SMW in `/var/opt/cray/log/`*sessionid*`/nhc-`*YYYYMMDD* with `'<node_health:`*M*`.`*m*`>'` in the message, where *M* is the major and *m* is the minor NHC revision number. All NHC messages are visible in the console file.

NHC prints a summary message per node at the end of Normal Mode and Suspect Mode when at least one test has failed on a node. For example:

```
<node_health:3.1> APID:100 (xtnhc) FAILURES: The following tests have failed in normal mode:
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Apinit_Ping
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Plugin /example/plugin
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Log Only ) Filesystem_Test on /mydir
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Free_Memory_Check
<node_health:3.1> APID:100 (xtnhc) FAILURES: End of list of 5 failed test(s)
```

The xtcheckhealth error and warning messages include node IDs and application IDs and are written to the console file on the SMW; for example:

```
[2010-04-05 23:07:09][c1-0c2s0n0]<node_health:3.0> APID:2773749
(check_apid) WARNING: Failure: File /dev/cpuset/2773749/tasks exists and is not empty. \
The following processes are running under expired APID
2773749:
[2010-04-05 23:07:09][c1-0c2s0n1]<node_health:3.0> APID:2773749
(check_apid) WARNING: Pid:   300 Name:           (marys_program) State: D
```

The xtcleanup_after script writes its normal launch information to the /var/log/xtcheckhealth_log file, which resides on the login nodes. The xtcleanup_after launch information includes the time that xtcleanup_after was launched and the xtcleanup_after's call to xtcheckhealth.

The xtcleanup_after script writes error output (launch failure information) to the /var/log/xtcheckhealth_log file, to the console file on the SMW, and to the syslog.

Example xtcleanup_after output follows:

```
Thu Apr 22 17:48:18 CDT 2010 <node_health> (xtcleanup_after) \
/opt/cray/nodehealth/3.0-1.0000.20840.30.8.ss/bin/xtcheckhealth -a 10515 \
-e 1 /tmp/apsysLVNqO9 /etc/opt/cray/nodehealth/nodehealth.conf
```

6.12.5 What If a Login Node Crashes While xtcheckhealth Binaries Are Monitoring Nodes?

If a login node crashes while some xtcheckhealth binaries on that login node are monitoring compute nodes that are in suspect state, those xtcheckhealth binaries will die when the login node crashes. When the login node that crashed is rebooted, a recovery action takes place. When the login node boots, the node_health_recovery binary starts up. This script checks for all compute nodes that are in suspect state and were last set to suspect state by this login node. The script then determines the APID of the application that was running on each of these compute nodes at the time of the crash. The script then launches an xtcheckhealth binary to monitor each of these compute nodes. One xtcheckhealth binary is launched per compute node monitored.

xtcheckhealth will be launched with this APID, so it can test for any processes that may have been left behind by that application. This testing only takes place if the Application_Exited_Check test is enabled in the configuration file. (The Application_Exited_Check test is enabled in the default NHC configuration file.) If the Application_Exited_Check test is not enabled, when the recovery action takes place, NHC does not run the Application_Exited_Check test and will not check for leftover processes. However, it will run any other NHC tests that are enabled in the configuration file.

Nodes will be changed from `suspect` state to `up` or `admindown`, depending upon whether they fail any health checks. No system administrator intervention should be necessary.

NHC automatically recovers the nodes in `suspect` state when the crashed login node is rebooted because the recovery feature runs on the rebooted login node. If the crashed login node is not rebooted, then manual intervention is required to rescue the nodes from `suspect` state. This manual recovery can commence as soon as the login node has crashed. To recover from a login node crash during the case in which a login node will not be rebooted, the `nhc_recovery` binary is provided to help you release the compute nodes owned by the crashed login node; see Procedure 36 on page 178. Also, see the `nhc_recovery`(8) man page for a description of the `nhc_recovery` binary usage.

Procedure 36. Recovering from a login node crash when a login node will not be rebooted

1. Create a *nodelistfile* that contains a list of the nodes in the system that are currently in Suspect Mode. The file must be a list of NIDs, one per line; do not include a blank line at the end of the file.

2. To list all of the `suspect` nodes in the system and which login nodes own those nodes, execute the following command; use the *nodelistfile* you created in step 1.

   `nhc_recovery -d` *`nodelistfile`*

3. Parse the `nhc_recovery` output for the NID of the login node that crashed. The file (for example, name it `nodelistfile_computenodes`) of this parsed list should contain all of the compute nodes owned by the crashed login node.

4. If you plan to recover the `suspect` nodes by using the option in step 6.a below, then complete this step; otherwise, skip this step.

   Note: This recovery method is recommended.

   From the list you created in step 3, create *nodelistfile*s containing nodes that share the same APID to determine the nodes from the crashed login node. For example, your *nodelistfile*s can be named `nodelistfile-APID1`, `nodelistfile-APID2`, `nodelistfile-APID3`, and so on.

5. Using the file you created in step 3, release all of the `suspect` compute nodes owned by the crashed login node. Execute the following command:

   `nhc_recovery -r` *`nodelistfile_computenodes`*

6. All of these compute nodes have been released in the database. However, they are all still in `suspect` state. Determine what to do with these `suspect` nodes from the following three options:

   a. (Cray recommends this option) Rerun NHC on a non-crashed login node to recover the nodes listed in step 4. Invoke NHC for each *nodelistfile*. Supply

as the APID argument the APID that corresponds to the *nodelistfile*; an iteration count of `0` (zero), which is the value normally supplied to NHC by ALPS; and an application exit code of `1` (one). An exit code of `1` ensures that NHC will run regardless of the value of the `runtests` variable (`always` or `errors`) in the NHC configuration file. For example:

```
xtcleanup_after -s nodelistfile-APID1  APID1  0  1
xtcleanup_after -s nodelistfile-APID2  APID2  0  1
xtcleanup_after -s nodelistfile-APID3  APID3  0  1
.
.
.
```

b. These `suspect` nodes can be set to `admindown` and their fate determined by further analysis.

c. These `suspect` nodes can be set back to `up`, but they were in Suspect Mode for a reason.

6.12.6 Disabling NHC

To disable NHC entirely, set the value of the `nhcon` global variable in the `/etc/opt/cray/nodehealth/nodehealth.conf` file to `off` (the default value in the file provided from Cray Inc. is `on`).

6.12.7 `nodehealth` Modulefile

To gain access to the NHC functions, the `nodehealth` module must be loaded. The `admin-modules` modulefile loads the `nodehealth` module, or you can load the `nodehealth` module by executing the following command:

```
module load nodehealth
```

The `Base-opts.default.local` file includes the `admin-modules` modulefile. For additional information about the `Base-opts.default.local` file, see System-wide Default Modulefiles on page 123.

6.12.8 Configuring the Node Health Checker to Use SSL

Note: NHC is configured to use secure sockets layer (SSL) protocol by default. Although this setting is configurable, Cray recommends that all sites configure NHC to use SSL.

If your site requires authentication and authorization to protect access to compute nodes, you can configure compute nodes to perform node health checking by using the `openssl` utility and secure sockets layer (SSL) protocol. SSL provides optional security functionality for NHC.

To enable the use of SSL, set the `NHC_SSL` setting in the `CLEinstall.conf` file to `yes`.

For more information about configuring NHC to use SSL, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

6.13 Activating Process Accounting for Service Nodes

The GNU 6.4 accounting package uses Berkeley Software Design (BSD) type process accounting. The GNU 6.4 process accounting is enabled for the Cray system's service nodes. The package name is `acct`; it can be activated using the `acct` boot script. To enable the `acct` boot script, execute the following command on the boot node root and/or shared root:

```
boot:~ # chkconfig acct on
```

The GNU 6.4 process accounting utilities process V2 and V3 format records seamlessly, even if the data is written to the same file. Output goes to an accounting file, which by default is `/var/account/pacct`. The accounting utilities provided for administration use are: `ac`, `lastcomm`, `accton`, and `sa`. The related man pages are accessible by using the `man` command.

6.14 Configuring Failover for Boot and SDB Nodes

The boot node is integral to the operation of a Cray system. Critical services like the Application Level Placement Scheduler (ALPS) and Lustre rely on the SDB and will fail if the SDB node is unavailable. The CLE release provides functionality to create standby boot and SDB nodes that automatically act as a backup in the event of primary node failure. Failover allows the system to keep running without an interrupt to the system or system services.

Note: The boot-node and SDB node failover features do not provide a failback capability.

A virtual network is configured for the boot and SDB nodes to support failover for these nodes. The virtual network is configured by default, regardless of the boot or SDB node failover configuration on your system.

The `CLEinstall` program provides the capability to change the default virtual network configuration, however, the default values are acceptable is most cases. For more information, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) or the `CLEinstall.conf`(5) man page.

6.14.1 Configuring Boot-node Failover

When you configure a secondary (backup) boot node, boot-node failover occurs automatically when the primary boot node fails.

The following services run on the boot node:

- NFS shared root (read-only)
- NFS persistent `/var` (read-write)
- Boot node daemon, `bnd`
- Hardware supervisory system (HSS) and system database (SDB) synchronization daemon, `xtdbsyncd`
- ALPS daemons `apbridge`, `apres`, and `apwatch` (for information about configuring ALPS, see Chapter 8, Using the Application Level Placement Scheduler (ALPS) on page 261)

When the primary boot node is booted, the backup boot node also begins to boot. However, the backup boot node makes a call to the `rca-helper` utility before it mounts its root file system, causing the backup boot node to be suspended until a primary boot-node failure event is detected.

The `rca-helper` daemon running on the backup boot node waits for a primary boot-node failure event, `ec_node_failed`. When the heartbeat of the primary boot node stops, the blade controller begins the heartbeat checking algorithm to determine if the primary boot node has failed. When the blade controller determines that the primary boot node has failed, it sends an `ec_heartbeat_stop` event to set the alert flag for the primary node. The primary boot node is halted through STONITH. Setting the `alert` flag on the node triggers the HSS state manager on the SMW to send out the `ec_node_failed event`.

When the `rca-helper` daemon running on the backup boot node receives an `ec_node_failed` event alerting it that the primary boot node has failed, it allows the boot process of the backup boot node to continue. Any remaining boot actions occur on the backup boot node. Booting of the backup boot node takes approximately two minutes.

Each service node runs a failover manager daemon (`rca_arpd`). When each service node's `rca_arpd` receives the `ec_node_failed` event, it takes appropriate action. The `rca_arpd` process updates the ARP cache entry for the boot node virtual IP address to reference the backup boot node.

The purpose of this implementation of boot-node failover is to ensure that the system continues running, not to guarantee that every job will continue running. Therefore, note the following:

- During the time the primary boot node has failed, any service node that tries to access its root file system will be I/O blocked until the backup boot node is online, at which time the request will be satisfied and the operation will resume. In general, this means if an application is running on a service node, it can continue to run if the application is in memory and does not need to access disk. If it attempts to access disk for any reason, it will be blocked until the backup boot node is online.
- Applications running on compute nodes are affected only if they cause a service node to access its root file system, in which case the service node function would be blocked until the backup boot node is online.

The following is a list of requirements for configuring your system for boot-node failover:

- The backup boot node requires a Host Bus Adapter (HBA) card to communicate with the RAID.

  Note: You must configure the backup boot node in the same zone as the primary boot node.

- You must ensure that the boot RAID host port can see the desired LUNs; for DDN, use the host port mapping; for NetApp (formerly LSI and Engenio), use SANshare in the SANtricity® Storage Manager.
- The backup boot node also requires a Gigabit Ethernet card connected through a Gigabit Ethernet switch to the same port on the SMW as the primary boot node (typically port 4 of the SMW quad Ethernet card).
- You must enable the STONITH capability on the blade or module of the primary boot node in order to use the boot node failover feature. STONITH is a per blade setting and not a per node setting. Ensure that your primary boot node is located on a separate blade from services with conflicting STONITH requirements, such as Lustre.

Procedure 37. Configuring boot-node failover

Note: If you configured boot-node failover during your CLE software installation or upgrade (as documented in the *Installing and Configuring Cray Linux Environment (CLE) Software*, S–2444), this procedure is not needed.

Tip: Use the `nid2nic` or `rtr --system-map` commands to translate between node or NIC IDs and physical ID names.

1. As `crayadm` on the SMW, halt the primary and alternate boot nodes.

Warning: Verify that your system is shut down before you invoke the `xtcli halt` command.

```
crayadm@smw:~> xtcli halt primary_id, backup_id
```

2. Update the default boot configuration used by the boot manager to boot nodes by using the `xtcli` command:

```
crayadm@smw:~> xtcli boot_cfg update -b primary_id,
backup_id -i /bootimagedir/bootimage
```

Or

If you are using `/raw0`, use the following command:

```
crayadm@smw:~> xtcli boot_cfg update -i /raw0
```

If you are using partitions, use the following command to designate the primary boot node and the backup boot node:

```
crayadm@smw:~> xtcli part_cfg update pN -b primary_id,backup_id -i /bootimagedir/bootimage
```

Or

If you are using `/raw0`, use the following command:

```
crayadm@smw:~> xtcli part_cfg update pN -i /raw0
```

3. Update the `CLEinstall.conf` file to designate the primary and backup boot nodes so the file has the correct settings when you do your next upgrade.

4. Boot the boot node.

5. The STONITH capability must be enabled on the blade of the primary boot node in order to use the boot-node failover feature.

Caution: STONITH is a per blade setting, not a per node setting. You must ensure that your primary boot node is not assigned to a blade that hosts services with conflicting STONITH requirements, such as Lustre.

a. Use the `xtdaemonconfig` command to determine the current STONITH setting on your primary boot node. For example, if the primary boot node is `c0-0c0s0n1` located on blade `c0-0c0s0`, type this command:

Note: If you have a partitioned system, invoke these commands with the `--partition p`*n* option.

```
crayadm@smw:~> xtdaemonconfig c0-0c0s0 | grep stonith
c0-0c0s0: stonith=false
```

b. To enable STONITH on your primary boot node, execute the following command:

```
crayadm@smw:~> xtdaemonconfig c0-0c0s0 stonith=true
c0-0c0s0: stonith=true
The expected response was received.
```

c. The STONITH setting does not survive a power cycle. You can maintain the STONITH setting for the primary boot node by adding the following line to your boot automation file:

```
# boot bootnode:
lappend actions {crms_exec "xtdaemonconfig c0-0c0s0 stonith=true"}
```

6. Boot the system.

Procedure 38. Disabling boot-node failover

- To disable boot-node failover, type these commands; in this example procedure, the primary boot node is `c0-0c0s0n1` and the backup boot node is `c2-0c1s7n1`.

```
crayadm@smw:~> xtcli halt c0-0c0s0n1,c2-0c1s7n1
crayadm@smw:~> xtcli boot_cfg update -b c0-0c0s0n1,c0-0c0s0n1
crayadm@smw:~> xtdaemonconfig c0-0c0s0 stonith=false
```

6.14.2 Configuring SDB Node Failover

When you configure a secondary (backup) SDB node, SDB node failover occurs automatically when the primary SDB node fails.

The CLE implementation of SDB node failover includes installation configuration parameters that facilitate automatic configuration, a `chkconfig` service called `sdbfailover`, and a `sdbfailover.conf` configuration file for defining site-specific commands to invoke on the backup SDB node.

The backup SDB node uses `/etc` files that are class or node specialized for the primary SDB node and not for the backup node itself; the `/etc` files for the backup node will be identical to those that existed on the primary SDB node. The backup SDB node also mounts the NID-specific persistent `/var` file system of the primary SDB node. For example, if the SDB NID is 270 and the backup SDB NID is 273 then `/snv/270/var` gets mounted on the backup SDB node when it takes over instead of `/snv/273/var`.

The following list summarizes requirements to implement SDB node failover on your Cray system.

- Designate a service node to be the alternate or backup SDB node. The backup SDB node requires a Host Bus Adapter (HBA) card to communicate with the RAID. This backup node is dedicated and cannot be used for other service I/O functions.
- Enable the STONITH capability on the blade or module of the primary SDB node in order to use the SDB node failover feature. STONITH is a per blade setting and not a per node setting. Ensure that your primary SDB node is located on a separate blade from services with conflicting STONITH requirements, such as Lustre.
- Enable SDB node failover by setting the `sdbnode_failover` parameter to `yes` in the `CLEinstall.conf` file prior to running the `CLEinstall` program.

  When this parameter is used to configure SDB node failover, the `CLEinstall` program will verify and turn on `chkconfig` services and associated configuration files for `sdbfailover`.
- Specify the primary and backup SDB nodes in the boot configuration by using the `xtcli` command with the `boot_cfg update -d` options. For more information, see the `xtcli`(8) man page.
- (Optional) Populate `/etc/opt/cray/sdb/sdbfailover.conf` with site-specific commands.

  When a failover occurs, the backup SDB node invokes all commands listed in the `/etc/opt/cray/sdb/sdbfailover.conf` file. Include commands in this file that are normally invoked during system start-up through boot automation scripts. In a SDB node failover situation, these commands must be invoked on the new (backup) SDB node. For example, you may include commands to start batch system software (if not started through `chkconfig`) or commands to add a route to an external license server.

If at any time you reconfigure your system to use a different primary SDB node, you must enable STONITH for the new SDB node and disable STONITH for the previous node.

For procedures to configure SDB node failover during a CLE software installation, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

6.14.3 The Node ARP Management Daemon (`rca_arpd`)

The node ARP management daemon (`rca_arpd`) manages the system ARP cache. This daemon deletes the IP to hardware address (ARP) mappings for failed nodes and readds them when they become available. It will only manage ARP mappings on the high speed interconnect network and not external network interfaces such as Ethernet. If failover is configured, `rca_arpd` also manages ARP mappings for the backup boot or SDB node. When a node failed event from the primary boot or SDB node is received, `rca_arpd` updates the ARP mapping for the boot or SDB node virtual IP address to point to the backup node.

This functionality is included in the `cray-rca-compute` and `cray-rca-service` RPMs and is installed by default.

6.15 Creating Logical Machines

Logical Machines on page 61, introduces logical machines. Configure a logical machine (sometimes known as a *system partition*) with the `xtcli part_cfg` command.

Partition IDs are predefined as `p0` to `p31`. The default partition `p0` is reserved for the complete system and is no longer a valid ID once a system has been partitioned.

6.15.1 Creating Logical Machines on Cray XC30 Systems

Cray XC30 systems can have one or more cabinets. Systems with one or two compute cabinets scale at the blade level. For larger liquid-cooled Cray XC30 systems, every cabinet is fully populated (with 3 chassis), with the possible exception of the last cabinet.

For Cray XC30 systems, groups are made up of two-cabinet pairs starting from the beginning. The last group may not be completely full, and it can consist of 1 through 6 fully-populated chassis.

6.15.1.1 Multiple Group Systems

When a Cray XC30 system contains multiple groups, you may partition the system at a per-group level of granularity. Groups do not need to be sequentially positioned in a multi-group partition.

So, if a Cray XC30 system has more than 2 cabinets, every partition can consist of any number of groups; the last group (or remainder of system chassis that are not part of a full 6-chassis group) in the system should be considered a group whether it is fully-populated or not in this partitioning context.

6.15.1.2 Single Group, Multiple-chassis Systems

When a Cray XC30 system contains between two and six fully-populated chassis, then you may partition the system at a per-chassis level of granularity. Each partition must be at least one full chassis, and a chassis cannot be shared between partitions. Chassis do not need to be sequentially positioned in a multi-chassis partition.

6.15.1.3 Single Chassis Systems

When a Cray XC30 system is composed of a single fully-populated chassis, each slot must be in the same partition with its corresponding even/odd pair.

Note: Even/odd pair nodes (for example, slot 0 and slot 1, or slot 8 and slot 9) share optical connections and therefore must be in the same partition.

There are 16 slots (or blades) in a single chassis, making 8 even/odd slot pairs, and a maximum of 8 partitions. Single chassis systems can have any combination of even/odd slot pairs (for example: 4-4, 6-2, 4-2-2, 2-2-1-1-1-1, etc.) and even/odd slot pairs do not need to be sequentially positioned in a multiple slot pair partition. In order for a partition to be bootable, however, it must have a boot node, an SDB node, an I/O node, and a login node.

6.15.2 Configuring a Logical Machine

The logical machine can have one of three states:

- Empty — not configured
- Disabled — configured but not activated
- Enabled — configured and activated

When a partition is defined, its state changes to `DISABLED`. Undefined partitions are `EMPTY` by default.

Procedure 39. Configuring a logical machine

- Use the `xtcli part_cfg` command with the *part_cmd* option (`add` in the following example) to identify the operation to be performed and the *part_option* (`-m`, `-b`, `-d` and `-i`) to specify the characteristics of the logical machine. The boot image may be a raw device, such as `/raw0`, or a file.

Example 92. Creating a logical machine with a boot node and SDB node specifying the boot image path

```
crayadm@smw:~> xtcli part_cfg add p2 -m c0-0,c0-1,c0-2,c0-3 \
-b c0-0c0s0n0 -d c0-0c0s2n1 -i /bootimagedir/bootimage
```

Note: When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.

For the logical machine to be bootable, you must specify boot node and SDB node IDs.

For instructions on booting a logical machine, see Booting a Logical Machine on page 188.

For information about configuring boot-node failover, see Configuring Boot-node Failover on page 180.

To watch HSS events on the specified partition, execute the `xtconsumer -p` *partition_name* command.

To display the console text of the specified partition, execute the `xtconsole -p` *partition_name* command.

For more information, see the `xtcli_part`(8), `xtconsole`(8), and `xtconsumer`(8) man pages.

6.15.3 Booting a Logical Machine

The `xtbootsys --partition p`*N* option enables you to indicate which logical machine (partition) to boot. If you do not specify a partition name, the default partition `p0` (component name for the entire system) is booted. Alternatively, if you do not specify a partition name and you use the `CRMS_PARTITION` environment variable, this variable is used as the default partition name. Valid values are in the form `p`*#*, where *#* ranges from `0` to `31`.

Note: `xtbootsys` manages a link from `/var/opt/cray/log/`*partition*`-current` to the current *sessionid* directory for that partition, allowing you to be able to change to `/var/opt/cray/log/p1-current` for example.

To boot a partition, see Booting the System on page 69.

6.16 Updating Boot Configuration

The HSS `xtcli boot_cfg` command allow you to specify the primary and backup boot nodes and the primary and backup SDB nodes for `s0` or `p0` (the entire system).

Example 93. Updating boot configuration

Update the boot configuration using the boot image `/bootimagedir/bootimage`, primary boot node (for example, `c0-0c0s0n1`), backup boot node, primary SDB node, and the backup SDB node:

```
crayadm@smw:~> xtcli boot_cfg update -b primaryboot_id,backupboot_id \
-d primarySDB_id,backupSDB_id -i /bootimagedir/bootimage
```

For a partitioned system, use `xtcli part_cfg` to manage boot configurations for partitions. For more information, see the `xtcli_boot`(8) and `xtcli_part`(8) man pages.

For information about configuring failover, see Configuring Failover for Boot and SDB Nodes on page 180.

6.17 Modifying Boot Automation Files

Your boot automation files should be located in `/opt/cray/hss/default/etc` on the SMW. There are several automation files; for example, `auto.generic.cnl` and `auto.min.cnl`.

For boot automation scripts, the Lustre file system should start up before the compute nodes.

Note: You can also boot the system or shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file. For related procedures, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

If you use boot automation files, see the `xtbootsys`(8) man page, which provides detailed information about boot automation files, including descriptions of using the `xtbootsys crms_boot_loadfile` and `xtbootsys crms_boot_sdb_loadfile` automation file procedures.

6.18 Callout to `rc.local` During Boot

The file `/etc/init.d/rc.local` is available for local customization of the boot process. If this file/script is present, it is executed during the compute node boot process. This script is executed after `/init`, before any of the scripts in `/etc/init.d/rc3.d` and before `/etc/fstab` is processed.

6.19 Changing the System Software Version to be Booted

Release switching enables you to change between versions and releases of the CLE software that are installed concurrently on the system.

You must reboot the operating system to switch CLE releases on your Cray system. You cannot change a release while the mainframe is running. You must reboot each time you change versions; however, you do not need to reboot the SMW.

Minor release switching allows you to select one of the CLE software versions that are installed within a single system set and have the same base operating system release (for example, switching from 4.0.22, back to 4.0.21). Switching is achieved by modifying sets of symbolic links in the file system to refer to the requested release.

Major release switching requires that you have a separate set of disk partitions for each major operating system (for example, switching from 3.1.72, to 4.0.25). Each system set provides a complete set of all file system and boot images, thus making it possible to switch easily between two or more different versions of your CLE system software. Each system set can be an alternative location for an installation or upgrade of your Cray system. System sets are defined in the `/etc/sysset.conf` file on the SMW.

If multiple versions of the software are installed and no version is chosen, the most recently installed is used.

6.19.1 Minor Release Switching Within a System Set

The `xtrelswitch` command performs release switching by manipulating symbolic links in the file system and by setting the default version of modulefiles that are loaded at login. `xtrelswitch` uses a release version that is provided either in the `/etc/opt/cray/release/xtrelease` file or by the `xtrel=` boot parameter. If the latter is not provided, the former is used. The `xtrelswitch` command is not intended to be invoked interactively; rather it is called by other scripts as part of the boot sequence. Specifically, when the boot node is booted, this command is invoked to switch the components in the boot node and shared root file systems.

To accomplish minor release switching, you must set the `bootimage_xtrel` parameter to `yes` in your `CLEinstall.conf` installation configuration file. This will include the release version in your boot image parameters file. If you routinely switch between minor levels, you may find it more convenient to use a *bootimage* in */bootimagedir* (the boot image must be in the same path for both the SMW and the boot root), instead of the updating the `BOOT_IMAGE` disk partition.

Note: The `xtrelswitch` command does not support switching between major release levels, for example from CLE 4.0 to CLE 5.0.

For additional information, see the `xtrelswitch`(8) man page.

6.19.2 Major Release Switching Using Separate System Sets

When you use system sets to change the Cray software booted on your Cray system, you boot an entirely different file system. The switched components include:

- The boot node root file system
- The shared-root file system
- The disk partition containing the SDB
- The `syslog`, `ufs`, and persistent `/var` file systems

Booting a system set requires:

- The `/etc/sysset.conf` file that describes the available system sets.
- Choosing which boot image will be used for the next boot. Each system set label has at least one `BOOT_IMAGE`.
- Activating a boot image for the chosen system set label.

The `CLEinstall` program installs or upgrades a system set to a set of disk partitions on the Boot RAID. For more information about the `CLEinstall` program and the `/etc/sysset.conf` file, see the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) and the `sysset.conf`(5) man page.

Procedure 40. Booting a system set

1. Choose which system set in the `/etc/sysset.conf` file should be used for the boot. For example:

   ```
   LABEL:BLUE
   DESCRIPTION:BLUE system with production
   ```

2. For the chosen system set, there is at least one `BOOT_IMAGE` in the `/etc/sysset.conf` file. Look at the `/etc/sysset.conf` file to determine which boot image is associated with which raw device. For example, to get the `SMWdevice` entry for `BOOT_IMAGE0` for the chosen system set:

```
# function    SMWdevice   host   hostdevice  mountpoint  shared
 BOOT_IMAGE0 /dev/disk/by-id/scsi-3600a0b800026e1400000192c4b66eb70-part2 boot \
             /dev/disk/by-id/scsi-3600a0b800026e1400000192c4b66eb70-part2 /raw0 no
```

3. Set the next boot to use the boot image `BOOT_IMAGE0` from the `BLUE` system set, which is the `/dev/disk/by-id/scsi-3600a0b800026e1400000192c4b66eb70-part2` disk partition. There will be a link from `/raw0` to `/dev/disk/by-id/scsi-3600a0b800026e1400000192c4b66eb70-part2`.

   ```
   smw:~ # xtcli boot_cfg update -i /raw0
   ```

   Or, if you are working with a partitioned system, p*N*:

   ```
   smw:~ # xtcli part_cfg update pN -i /raw0
   ```

6.20 Changing the Service Database (SDB)

The SDB, which is a MySQL database, contains the XTAdmin system database. The XTAdmin database contains both persistent and nonpersistent tables. The `processor` and `service_processor` tables are nonpersistent and are created from the HSS data at boot time. The XTAdmin database tables track system configuration information. The SDB makes the system configuration information available to the Application Level Placement Scheduler (ALPS), which interacts with individual compute nodes running CNL.

Cray provides commands (see Updating Database Tables on page 193) that enable you to examine values in the SDB tables and update them when your system configuration changes.

Caution: Do not use MySQL commands to change table values directly. Doing so can leave the database in an inconsistent state.

Accounts that access MySQL by default contain a `.my.cnf` file in their home directories.

6.20.1 Service Database Tables

Table 5 describes the SDB tables, which belong to the XTAdmin database.

Table 5. Service Database Tables

Table Name	Function
`attributes`	Stores compute node attribute information
`lustre_failover`	Updates the database when a node's Lustre failover configuration changes
`lustre_service`	Updates the database when a node's Lustre service configuration changes
`filesystem`	Updates the database when a Lustre file system's configuration changes
`gpus`	Stores accelerator module (GPU) information
`processor`	Stores master list of processing elements and their status
`segment`	For nodes with multiple NUMA nodes, stores attribute information about the compute node and its associated NUMA nodes
`service_cmd`	Stores characteristics of a service
`service_config`	Stores processing element services that the resiliency communication agent (RCA) starts
`service_processor`	Stores nodes and classes (boot, login, server, I/O, or network)
`version`	Stores the database schema version

6.20.2 Database Security

Access to MySQL databases requires a user name and password. The MySQL accounts and privileges are shown in Table 6. For security purposes, Cray recommends changing the account passwords on a regular basis. Default MySQL account passwords and an example of how to change them are documented in *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444). To change the default MySQL passwords, also see Changing Default MySQL Passwords on the SDB on page 114.

Table 6. Database Privileges

Account	Privilege
MySQL basic	Read access to most tables; most applications use this account.
MySQL sys_mgmt	Most privileged; access to all information and commands.

6.20.3 Updating Database Tables

The CLE command pairs shown in Table 7 enable you to update tables in the SDB. One command converts the data into an ASCII text file that you can edit; the other writes the data back into the database file.

Table 7. Service Database Update Commands

Get Command	Put Command	Table Accessed	Reason to Use	Default File
`xtdb2proc`	`xtproc2db`	`processor`	Updates the database when a node is taken out of service	`./processor`
`xtdb2attr`	`xtattr2db`	`attributes`	Updates the database when node attributes change (see Setting and Viewing Node Attributes on page 201)	`./attribute`
`xtdb2nodeclasses`	`xtnodeclasses2db`	`service_processor`	Updates the database when a node's class changes (see	`./node_classes`

Get Command	Put Command	Table Accessed	Reason to Use	Default File
			Changing Nodes and Classes on page 195)	
xtdb2segment	xtsegment2db	segment	For nodes with multiple NUMA nodes, updates the database when attribute information about node changes (see Using the XTAdmin Database segment Table on page 205)	./segment
xtdb2servcmd	xtservcmd2db	service_cmd	Updates the database when characteristics of a service change	./serv_cmd
xtdb2servconfig	xtservconfig2db	service_config	Updates the database when services change	./serv_config
xtdb2etchosts	none	processor	Manages IP mapping for service nodes	none
xtdb2lustrefailover	xtlustrefailover2db	lustre_failover	Updates the database when a node's Lustre failover state changes	./lustre_failover
xtdb2lustreserv	xtlustreserv2db	lustre_service	Updates the database when a file system's failover process is changed	./lustre_serv
xtdb2filesys	xtfilesys2db	filesystem	Updates the database when a file system's status changes	./filesys

Get Command	Put Command	Table Accessed	Reason to Use	Default File
xtdb2gpus	xtgpus2db	gpus	Updates the database when attributes about the accelerators change	./gpus
xtprocadmin	none	processor	Displays or sets the current value of processor flags and node attributes in the service database (SDB). The batch scheduler and ALPS are impacted by changes to these flags and attributes.	none
xtservconfig	none	service_config	Adds, removes, or modifies service configuration in the SDB service_config table	none

6.20.3.1 Changing Nodes and Classes

The service_processor table tracks node IDs (NIDs) and their classes (see Class Name on page 58). The table is populated from the /etc/opt/cray/sdb/node_classes file on the boot node every time the system boots. Change this file to update the database when the classes of nodes change, for example, when you are adding login nodes.

Note: If you make changes to /etc/opt/cray/sdb/node_classes, you **must** make the same changes to the node class settings in CLEinstall.conf before performing an update or upgrade installation; otherwise, the install utility will complain about the inconsistency.

Note: The xtnodeclasses2db command inserts the node-class list into the database. It does not make any changes to the shared root. To change the shared root, invoke the xtnce command (see Changing the Class of a Node on page 141).

For more information, see the xtdb2nodeclasses(8) and xtnodeclasses2db(8) man pages.

6.21 Viewing the Service Database Contents with MySQL Commands

The service database is configured as part of the system installation (see the *Installing and Configuring Cray Linux Environment (CLE) Software*, S–2444).

Caution: Use MySQL commands to examine tables, but do not use them to change table values directly. Doing so can leave the database in an inconsistent state.

Procedure 41. Examining the service databases with MySQL commands

1. As user `crayadm`, on the SDB node, enter the MySQL shell.

```
crayadm@sdb:~> mysql -u basic -p
Enter password: ***********
mysql> show databases;
+-----------+
| Database  |
+-----------+
| XTAdmin   |
1-----------+
1 row in set (0.04 sec)
```

2. Select the `XTAdmin` database.

```
mysql> use XTAdmin;
Database changed
```

3. Display the tables in the `XTAdmin` database.

```
mysql> show tables;
+-------------------+
| Tables_in_XTAdmin |
+-------------------+
| attributes        |
| filesystem        |
| gpus              |
| lustre_failover   |
| lustre_service    |
| processor         |
| segment           |
| service_cmd       |
| service_config    |
| service_processor |
| version           |
+-------------------+
10 rows in set (0.00 sec)
```

4. Display the format of the `service_processor` table.

```
mysql> describe service_processor;
+------------+------------------+------+-----+--------+
 Field       | Type           |Null|Key|Default| Extra|
+-------------+------------------+------+-----+------+
|processor_id|int(10) unsigned|    |PRI|0      |      |
|service_type|varchar(64)     |YES |   |NULL   |      |
+-------------+------------------+------+-----+------+
2 rows in set (0.00 sec)
```

5. Display the contents of all fields in the `service_processor` table.

```
mysql> select * from XTAdmin.service_processor;
+--------------+--------------+
| processor_id | service_type |
+--------------+--------------+
|            0 | service      |
|            3 | service      |
|            4 | service      |
|            7 | service      |
|            8 | service      |
|           11 | service      |
|           12 | service      |
|           15 | service      |
|           16 | service      |
|           19 | service      |
|           20 | service      |
|           23 | service      |
|           24 | service      |
|           27 | service      |
+--------------+--------------+
14 rows in set (0.00 sec)
```

6. Display `processor_id` values from the `processor` table.

```
mysql> select processor_id from processor;
+--------------+
| processor_id |
+--------------+
|            0 |
|            3 |
|            4 |
|            7 |
|            8 |
|          103 |
|          104 |
|          107 |

    ...

|          192 |
|          195 |
+--------------+
162 rows in set (0.00 sec)
```

6.22 Configuring the Lustre File System

For a description of the Lustre file system and how to configure it, see the Lustre documentation at https://wiki.hpdd.intel.com/display/PUB/Documentation and your specific Lustre server documentation.

6.23 Exporting Lustre with NFSv3

Cray supports exporting a direct-attached Lustre file system with NFSv3 (NFSv4 support is deferred at this time). This feature allows hosts that are external to the Cray system to access direct-attached Lustre file systems that would otherwise not be available and it provides access to data stored on a Lustre file system to hosts that do not support a Lustre client. A service node Lustre client (with external connectivity such as Ethernet or InfiniBand™) can mount the Lustre file system and export that mounted file system via standard NFS methods.

Procedure 42. Configuring the NFS server for Lustre export

1. Enter `xtopview` for the node view of your service node Lustre client.

   ```
   boot:~ # xtopview -n 8
   node/8:/ #
   ```

2. Check the file specialization of the `/etc/exports` file with `xthowspec`.

   ```
   node/8:/ # xthowspec /etc/exports
   node:8:/etc/exports:node
   ```

   Note: The `/etc/exports` file should be specialized by node. For more information on specializing files, see the `xtspec`(8) man page.

3. (Optional) If `/etc/exports` is not node specialized, specialize the file for the node.

   ```
   node/8:/ # xtspec -n 8 /etc/exports
   ```

4. Edit the `/etc/exports` file and add an entry for the mounted Lustre file system you wish to export via NFS. Specifying `insecure` gives the best interoperability with the universe of potential client systems. The `fsid=`*value* setting is required to successfully export a Lustre file system. This value is a 32-bit integer assigned by the administrator that identifies the file system to NFS. For more information on NFS mount options, see the `mount`(8) and `nfs`(5) man pages.

```
node/8:/ # vi /etc/exports
/lus/nid00023    *(rw,insecure,no_root_squash,no_subtree_check,fsid=value)
```

Note: You can identify the mounted Lustre file systems on a node with the following command:

```
crayadm@nid00008:~> mount -t lustre
23@gni1:/lus_hera on /lus/nid00023 type lustre (rw,relatime,flock)
```

5. Repeat step 2 through step 3 for the `/etc/sysconfig/nfs` file so that it is node specialized.

6. Edit the `/etc/sysconfig/nfs` file and change the following parameter to disable NFSv4.

Important: NFSv4 is not supported for exporting Lustre at this time.

```
node/8:/ # vi /etc/sysconfig/nfs
# Enable NFSv4 support (yes/no)
#
NFS4_SUPPORT="no"
```

7. Configure the `nfsserver` and `rpcbind` services to start at boot with the `chkconfig` command.

```
node/8:/ # chkconfig nfsserver on
nfsserver  on
node/8:/ # chkconfig rpcbind on
rpcbind  on
```

8. Restart the `nfsserver` and `rpcbind` services to pick up the configuration changes made earlier.

```
node/8:/ # service nfsserver restart
Shutting down kernel based NFS server: nfsd                          done
Starting kernel based NFS server: mountd statd nfsd sm-notify        done
node/8:/ # service rpcbind restart
Shutting down rpcbind                                                done
Starting rpcbind                                                     done
```

9. Exit `xtopview`.

Procedure 43. Configuring the NFS client to mount the exported Lustre file system

Depending on your client system, your configuration may be different. This procedure contains general information that will help you configure your client system to properly mount the exported Lustre file system. Consult your client system documentation for specific configuration instructions.

1. As `root`, verify that the `nfs` client service is started at boot.
2. Add a line to the `/etc/fstab` file to mount the exported file system. The list below describes various recommended file system mount options. For more information on NFS mount options, see the `mount`(8) and `nfs`(5) man pages.

server@network`:`*/filesystem /client/mount/point* `lustre` *file_system_options* `0 0`

Recommended file system mount options

`rsize=1048576,wsize=1048576`

Set the read and write buffer sizes from the server at 1MiB. These options match the NFS read/write transaction to the Lustre filesystem block size, which reduces cache/buffer thrashing on the service node providing the NFS server functionality.

`soft,intr` Use a soft interruptible mount request.

`async` Use asynchronous NFS I/O. Once the NFS server has acknowledged receipt of an operation, let the NFS client move along even though the physical write to disk on the NFS server has not been confirmed. For sites that need end-to-end write-commit validation, set this option to `sync` instead.

`proto=tcp` Force use of TCP transport; this makes the larger `rsize`/`wsize` operations more efficient. This option reduces the potential for UDP retransmit occurrences, which improves end-to-end performance.

`relatime,timeo=600,local_lock=none`

Lock and time stamp handling, transaction timeout at 10 minutes.

`nfsvers=3` Use NFSv3 specifically. NFSv4 is not supported at this time.

3. Mount the file system manually or reboot the client to verify that it mounts correctly at boot.

6.24 Enabling File-locking for Lustre Clients

To enable file-locking for all Linux clients when mounting the Lustre file system on service nodes or on compute nodes, you must use the `flock` option for `mount`. For more information, see the Lustre documentation at http://wiki.whamcloud.com/display/PUB/Documentation..

Example 94. Sample mount line from compute node `/etc/fstab`

```
4@gni:136@gni:/filesystem  /lus/nid00004   lustre  rw,flock              0 0
```

6.25 Backing Up and Restoring Lustre Failover Tables

To minimize the potential impact of an event that creates data corruption in the SDB database, Cray recommends that you create a manual backup of the Lustre tables, which can be restored after a reinitialization of the SDB database.

Procedure 44. Manually backing up Lustre failover tables

1. Log on to the boot node as `root`.
2. Use the `mysqldump` command to back up the `lustre_service` table.

   ```
   boot# mysqldump -h sdb XTAdmin lustre_service > \
   /var/tmp/lustre_service.sql
   ```

3. Back up the `lustre_failover` table.

   ```
   boot# mysqldump -h sdb XTAdmin lustre_failover > \
   /var/tmp/lustre_failover.sql
   ```

Procedure 45. Manually restoring Lustre failover tables

1. Log on to the boot node as `root`.
2. After the SDB database is recreated, use the `mysqldump` command to restore the `lustre_service` table.

   ```
   boot# mysqldump -h sdb XTAdmin < \
   /var/tmp/lustre_service.sql
   ```

3. Restore the `lustre_failover` table.

   ```
   boot# mysqldump -h sdb XTAdmin < /var/tmp/lustre_failover.sql
   ```

6.26 Configuring Cray Data Virtualization Service (Cray DVS)

For a description of the Cray DVS parallel I/O forwarding service and how to configure it, see *Introduction to Cray Data Virtualization Service* (S–0005).

6.27 Setting and Viewing Node Attributes

Users can control the selection of the compute nodes on which to run their applications and can select nodes on the basis of desired characteristics (*node attributes*). This allows a placement scheduler to schedule jobs based on the node attributes.

A user invokes the `cnselect` command to specify node-selection criteria. The `cnselect` script uses these selection criteria to query the table of node attributes in the SDB and returns a node list to the user based on the results of the query.

When launching the application, the user includes the node list using the `aprun -L` *node_list* option as described on the `aprun`(1) man page. The ALPS placement scheduler allocates nodes based on this list.

Note: To meet specific user needs, you can modify the `cnselect` script. For additional information about the `cnselect` script, see the `cnselect`(1) man page.

6.27.1 Setting Node Attributes Using the /etc/opt/cray/sdb/attr.xthwinv.xml and /etc/opt/cray/sdb/attr.defaults Files

In order for users to select desired node attributes, you must first set the characteristics of individual compute nodes. Node attribute information is written to the `/etc/opt/cray/sdb/attributes` data file and loaded into the `attributes` table in the SDB when the SDB is booted.

6.27.1.1 Generating the /etc/opt/cray/sdb/attributes File

Data for the `/etc/opt/cray/sdb/attributes` file comes from two other files: the `/etc/opt/cray/sdb/attr.xthwinv.xml` file, which contains information to generate the hardware attributes for each node, and the `/etc/opt/cray/sdb/attr.defaults` file, which allows administrators to set values for specific nodes (or all nodes if a `DEFAULT` is specified). The `xtprocadmin`(8) man page includes a description of the attributes fields used by these two files.

- The `/etc/opt/cray/sdb/attr.xthwinv.xml` file is created by `CLEinstall` and automatically regenerated by `xtbootsys` at each boot through the `xthwinv -x` command.

  To manually generate the `/etc/opt/cray/sdb/attr.xthwinv.xml` file, invoke the `xthwinv -x` command on the System Management Workstation (SMW) through the boot node, redirecting the output to the `/etc/opt/cray/sdb/attr.xthwinv.xml` file on the boot node; for example:

  ```
  boot:~ # ssh smw xthwinv -x s0 > /etc/opt/cray/sdb/attr.xthwinv.xml
  ```

  For additional information about the `xthwinv` command, see the `xthwinv`(8) man page.

  Note: If you have blades powered down when you want to upgrade your software, see the `CLEinstall`(8) man page for instructions on using the `--xthwinvxmlfile` option during your upgrade process.

- The `/etc/opt/cray/sdb/attr.defaults`file can be used to set an attribute value on any node, but it is primarily used for assigning labels to nodes (see Example 95).

  `label0`
  `label1`
  `label2`
  `label3` — Each label is a string of up to 32 characters; the string cannot contain any spaces or shell-sensitive characters.

  These labels can be applied to all nodes or to a given set of nodes.

Note: Do **not** attempt to set hardware attributes (memory size, clock speed, and cores) in the `attr.defaults` file because the values will be overwritten by those already specified in the `/etc/opt/cray/sdb/attr.xthwinv.xml` file.

To create the `attr.defaults` file, copy the example file provided in `/opt/cray/sdb/default/etc/attr.defaults.example`. Edit the file to modify the existing attribute settings and to create site-specific attributes as needed. If you have run `CLEinstall` previously, `attr.defaults` was already copied and exists in that location.

In addition to the attributes in the `/etc/opt/cray/sdb/attr.defaults` file, there are two keywords that allow you to describe the node or set of nodes to which attributes are assigned. For global default-attribute values that apply to the entire system, the line that specifies an attribute must begin with the `DEFAULT:` keyword. For example:

```
DEFAULT:  osclass=2
```

The `nodeid` keyword assigns attributes to a specific node or set of nodes and overrides a default setting. For values that apply only to certain nodes, the line that specifies the attributes must begin with `nodeid=[`*RANGE*`]`, where *RANGE* is a comma-separated list of nodes and ranges that have the form *m*,*n* or *m*-*n*. For example:

```
nodeid=234,245-248 label3='GREEN'
```

Example 95. Using node attribute labels to assign nodes to user groups

The following example uses labels to assign groups of compute nodes to specific user groups without the need to partition the system:

```
nodeid=101-500 label0=physicsdept
nodeid=501-1000 label1=csdept
nodeid=50-100,1001 label2=biologydept
```

6.27.2 SDB `attributes` Table

When the SDB boots, it reads the `/etc/opt/cray/sdb/attributes` file and loads it into the SDB `attributes` table.

To display the format of the `attributes` SDB table, use the `mysql` command:

```
crayadm@login:~> mysql -e "desc attributes;" -h sdb XTAdmin
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| nodeid   | int(32) unsigned | NO   | PRI | 0       |       |
| archtype | int(4) unsigned  | NO   |     | 2       |       |
| osclass  | int(4) unsigned  | NO   |     | 2       |       |
| coremask | int(4) unsigned  | NO   |     | 1       |       |
| availmem | int(32) unsigned | NO   |     | 0       |       |
| pageszl2 | int(32) unsigned | NO   |     | 12      |       |
| clockmhz | int(32) unsigned | YES  |     | NULL    |       |
| label0   | varchar(32)      | YES  |     | NULL    |       |
| label1   | varchar(32)      | YES  |     | NULL    |       |
| label2   | varchar(32)      | YES  |     | NULL    |       |
| label3   | varchar(32)      | YES  |     | NULL    |       |
| numcores | int(4) unsigned  | NO   |     | 1       |       |
| sockets  | int(4) unsigned  | NO   |     | 1       |       |
| dies     | int(4) unsigned  | NO   |     | 1       |       |
+----------+------------------+------+-----+---------+-------+
```

The service database command pair `xtdb2attr` and `xtattr2db` enables you to update the `attributes` table in the SDB. For additional information about updating SDB tables using command pairs, see Updating Database Tables on page 193.

6.27.3 Setting Attributes Using the `xtprocadmin` Command

You can use the `xtprocadmin -a` *attr=value* command to temporarily set certain site-specific attributes. Using the `xtprocadmin -a` *attr=value* command to set certain site-specific attributes is **not** persistent across reboots. Attribute settings that are intended to be persistent across reboots (such as labels) must be specified in the `attr.defaults` file.

Note: For compute nodes, `xtprocadmin` changes to attributes requires that you restart the `apbridge` daemon on the boot node in order for ALPS to detect changes that the `xtprocadmin` command has made to the SDB. Restarting the other ALPS components (for example, on the SDB node or on the login node if they are separate nodes) is not necessary. To restart `apbridge`, log into the boot node as `root` and execute the following command:

```
boot:~ # /etc/init.d/alps restart
```

For example, the following command creates a new `label1` attribute value for the compute node whose NID is 350; you must be user `root` and execute the `xtprocadmin` command from a service node, and the SDB must be running:

```
boot:~ # xtprocadmin -n 350 -a label1=eedept
```

The output is:

```
Connected
NID    (HEX)    NODENAME    TYPE     LABEL1
350    0x15e    c1-0c1s0n0  compute  eedept
```

Then restart the `apbridge` daemon on the boot node in order for ALPS to detect changes that the `xtprocadmin` command has made to the SDB.

```
boot:~ # /etc/init.d/alps restart
```

6.27.4 Viewing Node Attributes

Use the `xtprocadmin` command to view current node attributes. The `xtprocadmin -A` option lists all attributes of selected nodes. The `xtprocadmin -a` *attr1,attr2* option lists selected attributes of selected nodes.

6.28 Using the XTAdmin Database `segment` Table

The XTAdmin database contains a `segment` table that supports the memory affinity optimization tools for applications and CPU affinity options for all Cray compute nodes. The CPU affinity options apply to all Cray multicore compute nodes.

The `segment` table is similar to the `attributes` table but differs in that a node may have multiple segments associated with it; the `attributes` table provides summary information for each node.

In order to address the application launch and placement requirements for compute nodes with two or more NUMA nodes, the Application Level Placement Scheduler (ALPS) requires additional information that characterizes the intranode topology of the system. This data is stored in the `segment` table of the XTAdmin database and acquired by `apbridge` when ALPS is started, in much the same way that node attribute data is acquired. (For more information about XTAdmin database tables, see Changing the Service Database (SDB) on page 191.)

The segment table contains the following fields:

- node_id is the node identifier that maps to the nodeid field of the attributes table and processor_id field of the processor table.
- socket_id contains a unique ordinal for each processor socket.
- die_id contains a unique ordinal for each processor die; with this release, die_id is 0 in the segment table and is otherwise unused (reserved for future use).
- numcores is the number of integer cores per node; in systems with accelerators this only applies to the host processor (CPU).
- coremask is the processor core mask. The coremask has a bit set for each core of a CPU. 24-core nodes will have a value of 16777215 (hex 0xFFFFFF).

  Note: coremask is deprecated and will be removed in a future release.
- mempgs represents the amount of memory available, in Megabytes, to a single segment.

The /etc/sysconfig/xt file contains SDBSEG field, which specifies the location of the segment table file; by default, SDBSEG=/etc/opt/cray/sdb/segment.

To update the segment table, use the following service database commands:

- xtdb2segment, which converts the data into an ASCII text file that can be edited
- xtsegment2db, which writes the data back into the database file

For more information, see the xtdb2segment(8) and xtsegment2db(8) man pages.

After manually updating the segment table, you can log on to any login node or the SDB node as root and execute the apmgr resync command to request ALPS to reevaluate the configuration node segment information and update its information.

Note: If ALPS or any portion of the feature fails in relation to segment scheduling, ALPS reverts to the standard scheduling procedure.

6.29 Configuring Networking Services

6.29.1 Changing the High-speed Network (HSN)

To change your system interconnection network (HSN) address ranges, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

6.29.2 Network File System (NFS)

The Network File System (NFS) version 4 distributed file system protocol is supported. NFS is enabled by default on the `boot`, `sdb`, and `ufs` service nodes but is not enabled on compute nodes. Support for NFSv4 is included as part of the SLES software.

The CLE installation tool supports NFS tuning via `/etc/sysconfig/nfs` and `/etc/init.d/nfsserver` on the boot node. The `nfs_mountd_num_threads` and `use_kernel_nfsd_number` parameters in the `CLEinstall.conf` installation configuration file control an NFS `mountd` tuning parameter that is added to `/etc/sysconfig/nfs` and used by `/etc/init.d/nfsserver` to configure the number of `mountd` threads on the boot node. By default, NFS `mountd` behavior is a single thread. If you have a larger Cray system (greater than 50 service I/O nodes), contact your Cray service representative for assistance changing the default setting.

If you wish to enable the `nfsserver` service on all service nodes, you may do so by setting the `CLEinstall.conf` `nfsserver` parameter to `yes`. The default setting is `no`.

6.29.3 Configuring Ethernet Link Aggregation (Bonding, Channel Bonding)

Linux Ethernet link aggregation is generally used to increase aggregate bandwidth by combining multiple Ethernet channels into a single virtual channel. Bonding can also be used to increase the availability of a link by utilizing other interfaces in the bond when one of the links in that bond fails.

Procedure 46. Configuring an I/O service node bonding interface

1. On the boot node, run the `xtopview` command for the node that needs the bonding interface configured. For example, to access node 2, type the following:

   ```
   boot:~ # xtopview -n 2
   node/2:/ #
   ```

2. Create and specialize the following files: `/etc/sysconfig/network/ifcfg-bond0`, `/etc/sysconfig/network/ifcfg-eth0`, and `/etc/sysconfig/network/ifcfg-eth1`.

   ```
   node/2:/ # touch /etc/sysconfig/network/ifcfg-bond0
   node/2:/ # xtspec -n 2 /etc/sysconfig/network/ifcfg-bond0
   node/2:/ # touch /etc/sysconfig/network/ifcfg-eth0
   node/2:/ # xtspec -n 2 /etc/sysconfig/network/ifcfg-eth0
   node/2:/ # touch /etc/sysconfig/network/ifcfg-eth1
   node/2:/ # xtspec -n 2 /etc/sysconfig/network/ifcfg-eth1
   ```

3. Edit the previously created files to include your specific network settings.

```
node/2:/ # vi /etc/sysconfig/network/ifcfg-bond0
BOOTPROTO="static"
BROADCAST="10.0.2.255"
IPADDR="10.0.2.10"
NETMASK="255.255.0.0"
NETWORK="10.0.2.0"
REMOTE_IPADDR=""
STARTMODE="onboot"
BONDING_MASTER="yes"
BONDING_MODULE_OPTS="mode=active-backup primary=eth1"
BONDING_SLAVE0="eth0"
BONDING_SLAVE1="eth1"

node/2:/ # vi /etc/sysconfig/network/ifcfg-eth0
BOOTPROTO='static'
STARTMODE='onboot'
MASTER=bond0
SLAVE=yes
REMOTE_IPADDR=''
IPV6INIT=no

node/2:/ # vi /etc/sysconfig/network/ifcfg-eth1
BOOTPROTO='static'
STARTMODE='onboot'
MASTER=bond0
SLAVE=yes
REMOTE_IPADDR=''
IPV6INIT=no
```

4. Exit from `xtopview`

```
node/2:/ # exit
```

For more information on Ethernet link aggregation, see the Linux documentation file `/usr/src/linux/Documentation/networking/bonding.txt`, installed on your system.

6.29.4 Configuring a Virtual Local Area Network (VLAN) Interface

This procedure configures an 802.1Q standard VLAN.

Procedure 47. Configuring a Virtual Local Area Network (VLAN) interface

1. On the boot node, run the `xtopview` command for the node that needs the VLAN configured. For example, to access node 2, type the following:

```
boot:~ # xtopview -n 2
   node/2:/ #
```

2. Create and specialize a file named `/etc/sysconfig/network/ifcfg-vlan`*N*, where *N* is the VLAN ID. The following example creates and specializes the `vlan2` file:

```
node/2:/ # touch /etc/sysconfig/network/ifcfg-vlan2
   node/2:/ # xtspec -n 2 /etc/sysconfig/network/ifcfg-vlan2
```

3. Edit the `/etc/sysconfig/network/ifcfg-vlan`*N* file to include your usual network settings. It must also include variable `ETHERDEVICE` that provides the real interface for the VLAN. The real interface will be set up automatically; it does not require a configuration file. For additional information, see the `ifcfg-vlan`(5) man page. The following example sets up `vlan2` on top of `eth0`:

   ```
   ifcfg-vlan2
        STARTMODE=onboot
        ETHERDEVICE=eth0
        IPADDR=192.168.3.27/24
   ```

   An interface named `vlan2` will be created when the system boots.

4. Exit from `xtopview`.

   ```
   node/2:/ # exit
   ```

6.29.5 Increasing Size of ARP Tables

To increase the size of ARP tables, change the `ARP_OVERHEAD` parameter in the `/etc/sysconfig/xt` file. `ARP_OVERHEAD` should be set to a value greater than the number of hosts in all locally attached external networks; the current default is `0`.

6.29.6 Configuring Realm-specific IP Addressing (RSIP)

Realm-Specific Internet Protocol (RSIP) enables internal client nodes, such as compute nodes, to reach external IP networking resources. Support for RSIP is available with CLE on systems that have CNL compute nodes.

Note: RSIP for IPv4 TCP and User Datagram Protocol (UDP) transport protocols are supported. Internet Protocol Security (IPSec) and IPv6 protocols are not supported.

RSIP is composed of two main components: RSIP clients and RSIP servers or gateways. You configure RSIP and select servers using RSIP parameters in `CLEinstall.conf`. By default, when RSIP is enabled, all CNL compute nodes are configured to be RSIP clients.

On your Cray system, RSIP servers must be service nodes with an external IP interface such as a 10-GbE network interface card (NIC). You can configure multiple RSIP servers using multiple service nodes, however only one RSIP daemon (`rsipd`) and one external interface is allowed per service node. Cray requires that you configure RSIP servers as dedicated network nodes.

Warning: Do not configure login nodes or service nodes that provide Lustre or batch services as RSIP servers. Failure to set up an RSIP server as a dedicated network node will disrupt network functionality.

The performance impact of configuring RSIP is negligible; very little noise is generated by the RSIP client. RSIP clients will issue a lease refresh message request/response pair once an hour, at a rate of 10 clients per second, but otherwise are largely silent.

To configure RSIP for your Cray system, first determine which service nodes and associated Ethernet devices will be used to provide RSIP services. Optionally, determine if you will configure service nodes with no external IP interfaces (isolated service nodes) to act as RSIP clients. After selecting RSIP servers based on your machine-specific networking hardware configuration, follow Procedure 48 on page 211 to complete a default RSIP configuration and setup.

Enhancements to the default RSIP configuration require a detailed analysis of specific site configuration and requirements. Contact your Cray representative for assistance in changing the default RSIP configuration.

6.29.6.1 Using the `CLEinstall` Program to Install and Configure RSIP

The `CLEinstall` program can be configured to automatically install RSIP either during a system software upgrade or as a separate event. In either case, you will need to update the compute node boot image and restart your Cray system before RSIP is functional.

When you set the following RSIP-specific parameters in the `CLEinstall.conf` file, `CLEinstall` will load the RSIP RPM, modify `rsipd.conf` and invoke the appropriate `xtrsipcfg` commands to configure RSIP for your system.

`rsip_nodes=`

Specifies the RSIP servers. Populate with the node IDs of the nodes you have identified as RSIP servers.

`rsip_interfaces=`

Specifies the IP interface for each RSIP server node. List the interfaces in the same order specified by the `rsip_nodes` parameter.

`rsip_servicenode_clients=`

Specifies a space separated integer list of service nodes you would like to use for RSIP clients.

Warning: Do not configure service nodes with external network connections as RSIP clients. Configuring a network node as an RSIP client will disrupt network functionality. Service nodes with external network connections will route all non-local traffic into the RSIP tunnel and IP may not function as desired.

`CNL_rsip=yes`

Enables the RSIP client on CNL compute nodes. Optionally, you can edit the `/var/opt/cray/install/shell_bootimage_`*LABEL*`.sh` script and set `CNL_RSIP=y`.

If you are configuring RSIP for the first time during an installation or upgrade of your CLE system software, follow RSIP-specific instructions in the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444). If you are configuring RSIP as a separate event, follow Procedure 48 on page 211. If you already configured RSIP and want to add isolated service nodes as RSIP clients, follow Procedure 49 on page 213.

For additional information about configuring RSIP, see the `xtrsipcfg`(8), `rsipd`(8), and `rsipd.conf`(5) man pages.

Procedure 48. Installing, configuring, and starting RSIP clients and servers

1. Edit `CLEinstall.conf` for your RSIP configuration. For example, to configure nodes 16 and 20 as RSIP servers with an external interface named `eth0` and node 64 as an RSIP server with an external interface named `eth1`; and node 0 as a service node RSIP client, make these changes.

   ```
   smw:~ # vi /home/crayadm/install.xtrelease/CLEinstall.conf
   rsip_nodes=16 20 64
   rsip_interfaces=eth0 eth0 eth1
   rsip_servicenode_clients=0
   CNL_rsip=yes
   ```

2. Invoke the `CLEinstall` program on the SMW; you **must** specify the *xtrelease* that is currently installed on the system set you are using and located in the `CLEmedia` directory.

```
smw:~ # /home/crayadm/install.xtrelease/CLEinstall --upgrade \
--label=system_set_label --XTrelease=xtrelease \
--configfile=/home/crayadm/install.xtrelease/CLEinstall.conf \
--CLEmedia=/home/crayadm/install.xtrelease
```

3. Type **y** and press the `Enter` key to proceed when prompted to update the boot root and again for the shared root.

   ```
   *** Do you wish to continue? (y/n) --> y
   ```

   Upon completion, `CLEinstall` lists suggested commands to finish the

installation. Those commands are also described here. For more information about running the `CLEinstall` program, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

4. Rebuild the boot image using `/var/opt/cray/install/shell_bootimage_`*LABEL*`.sh`, `xtbootimg` and `xtcli` commands. Suggested commands are included in output from `CLEinstall` and `shell_bootimage_`*LABEL*`.sh`. For more information about creating boot images, follow Procedure 2 on page 64.

5. Run the `shell_post_install.sh` script on the SMW to unmount the boot root and shared root file systems and perform other cleanup as necessary.

```
smw:~# /var/opt/cray/install/shell_post_install.sh /bootroot0 /sharedroot0
```

6. (Optional) If you are configuring a service node RSIP client, edit the boot automation file to start the RSIP client. On the isolated service node, invoke a `modprobe` of the `krsip` module with an IP argument pointing to the HSN IP address of an RSIP server node, and specifying the number of ports requested. For example, if the IP address of the RSIP server is `10.128.0.17`, the isolated service node is `nid00000`, and 32 ports are requested, make these changes.

```
crayadm@smw:~> vi /opt/cray/etc/auto.xthostname
After the line or lines that start the RSIP servers add:
# RSIP client startup
lappend actions { crms_exec_via_bootnode "nid00000" "root" "modprobe krsip
ip=10.128.0.17use_rsip_local_ports=1 num_ports=32" }
```

7. Boot your Cray system; for example:

```
crayadm@smw:~> xtbootsys -a auto.xthostname
```

   Note: RSIP clients on the compute nodes make connections to the RSIP server(s) during system boot. Initiation of these connections is staggered at a rate of 10 clients per second; during that time, connectivity over RSIP tunnels will be unreliable. Avoid using RSIP services for several minutes following a system boot; larger systems will require more time for connections to complete.

8. Test RSIP functionality. From a login node, log on to an RSIP client node (compute node) and ping the IP address of the SMW or other host external to your Cray system. For example, if *nid00074* is a compute node and *10.3.1.1* is a valid external IP address, type these commands.

```
crayadm@login:~> ssh root@nid00074
root@nid00074's password:
Welcome to the initramfs
# ping 10.3.1.1
10.3.1.1 is alive!
#
```

Procedure 49. Adding isolated service nodes as RSIP clients

You can configure service nodes that are isolated from the network as RSIP clients. This procedure assumes that RSIP is already configured and functional on your Cray system. If you have not installed and configured RSIP on your system, follow Procedure 48 on page 211, which includes an optional step to configure isolated service nodes as RSIP clients.

Warning: Do not configure service nodes with external network connections as RSIP clients. Configuring a network node as an RSIP client will disrupt network functionality; Service nodes with external network connections will route all non-local traffic into the RSIP tunnel and IP may not function as desired.

1. Select one of your RSIP servers to provide access for the isolated service node. In this example, we have chosen the RSIP server `nid00016`.

2. Log on to the boot node and invoke `xtopview` in the node view for the RSIP server you have selected; for example:

   ```
   boot:~ # xtopview -n 16
   node/16:/ #
   ```

   Modify `max_clients` in the `rsipd.conf` file to add an additional client for each isolated service node you are configuring. For example, if you configured 300 RSIP clients (compute nodes), change 300 to 301.

   ```
   node/16:/ # vi /etc/opt/cray/rsipd/rsipd.conf
   max_clients 301
   ```

3. Load the RSIP client on the node. On the isolated service node, invoke a `modprobe` of the `krsip` module with an IP argument pointing to the HSN IP address of the RSIP server node you selected in step 1. For example, if the IP address of the RSIP server is `10.128.0.17`, the isolated service node is `nid00023`, and 32 ports were requested, type these commands.

   ```
   boot:~ # ssh nid00023
   nid00023:~ #  modprobe krsip ip=10.128.0.17use_rsip_local_ports=1
   num_ports=32
   ```

4. Edit the boot automation file to start the RSIP client. Using the example from the previous steps, make these changes.

```
crayadm@smw:~> vi /opt/cray/etc/auto.xthostname
After the line or lines that start the RSIP servers add:
# RSIP client startup
lappend actions { crms_exec_via_bootnode "nid00023" "root" "modprobe krsip \
ip=10.128.0.17use_rsip_local_ports=1 num_ports=32" }
```

6.29.7 IP Routes for CNL Nodes in the `/etc/routes` File

You can edit the `/etc/routes` file in the compute node template image on the SMW to provide route entries for compute nodes. This provides a simple mechanism for you to configure routing access from compute nodes to login and network nodes using external IP destinations without having to traverse RSIP tunnels. This mechanism is not intended to be used for general-purpose routing of internal HSN IP traffic. It is intended only to provide IP routes for compute nodes that need to reach external IP addresses or external networks. A new `/etc/routes` file is created in the compute images and is examined during startup. Non-comment, non-blank lines are passed to the `route add` command. The empty template file provided contains comments describing the syntax.

6.30 Updating the System Configuration After a Blade Change

When a blade is changed in a Cray system, you need to update the configuration of the system. You will need to do this after:

- Adding additional blades to the system
- Removing a blade from your system configuration
- Changing a blade from a compute blade to a service blade
- Changing a blade from a service blade to a compute blade

You can update the system configuration when the system is not booted (see Updating the System Configuration When the System is Not Booted on page 214) or while the system is booted (see Updating the System Configuration While the System Is Booted on page 217).

6.30.1 Updating the System Configuration When the System is Not Booted

Important: After you have made hardware changes and you want to update the system configuration when the system is **not** booted, follow the procedures in this section.

Procedure 50. Updating the SMW configuration after hardware changes when the system is not booted

Note: When blades are changed that have the same blade type and Processor Daughter Card (PDC) type, `xtdiscover` does not need to be executed. If these blades are changed or if cabling changes are made and `xtdiscover` does not have to be executed, you must still execute the `xtbounce --linktune` command, which forces `xtbounce` to do full tuning on the system. For more information about the `xtbounce --linktune` command, see the `xtbounce`(8) man page.

1. Execute the `xtdiscover` command, which updates the system configuration to reflect the changed blade configuration.

2. If the blade or PDC type is different, execute the `xtdiscover` command.

   ```
   smw:~ # xtdiscover
   ```

3. If the blade or PDC type is different, you must execute the `xtzap --blade` command.

   ```
   smw:~ # xtzap --blade blade_cname
   ```

4. Execute the `xtbounce --linktune=all` command to tune PCIe and HSN links on the system. If using a partition and not the entire machine, use `p`*N* instead of `s0`.

   ```
   smw:~ # xtbounce --linktune=all s0
   ```

5. Capture the system configuration for `CLEinstall` by executing the following `xthwinv` commands. (The `CLEinstall --xthwinvxmlfile` option will order the `CLEinstall` program to use this captured configuration information.) If using a partition and not the entire machine, use `p`*N* instead of `s0`.

   ```
   smw:~ # xthwinv -x s0 > /home/crayadm/install.5.2.14/xthwinv.s0.xml
   ```

   For additional information about the `CLEinstall` and `xthwinv` commands, see the `CLEinstall`(8) and `xthwinv`(8) man pages.

Procedure 51. Using `CLEinstall` to update the system configuration after adding a blade to a system when the system is not booted

1. Update the `CLEinstall.conf` file with the blade changes.

2. Execute the `CLEinstall` command, including the `--xthwinvxmlfile` option, to update the system and prepare a boot image. The `CLEinstall --xthwinvxmlfile` option orders the `CLEinstall` program to use previously captured configuration information (see Procedure 50 on page 214). If using a partition and not the entire machine, use `p`*N* instead of `s0`.

   ```
   smw:~ # ./CLEinstall --label=LABEL --upgrade --XTrelease=5.2.14 \
   --xthwinvxmlfile=/home/crayadm/install.5.2.14/xthwinv.s0.xml \
   --configfile=/home/crayadm/install.5.2.14/CLEinstall.conf \
   --CLEmedia=/home/crayadm/install.5.2.14
   ```

3. Run the `shell_bootimage_`*LABEL*`.sh` script, where *LABEL* is the system set label specified in `/etc/sysset.conf` for this boot image. Specify the `-c` option to automatically create and set the boot image for the next boot. For example:

   ```
   smw:~ # /var/opt/cray/install/shell_bootimage_LABEL.sh -c
   ```

   For information about additional options accepted by this script, use the `-h` option to display a help message.

4. Run the `shell_post_install.sh` script on the SMW to unmount the boot root and shared root file systems and perform other cleanup as needed.

   ```
   smw:~ # /var/opt/cray/install/shell_post_install.sh
   /bootroot0 /sharedroot0
   ```

5. If you are adding a compute blade into the system configuration, boot the system as you normally boot your system.

   If you are adding a service blade into the system configuration, complete the following steps.

   a. Boot the boot node.

   b. If the login node(s) or RSIP node(s) have changed, edit `/etc/sysconfig/network/ifcfg-eth0`.

   c. Boot the SDB node and all other service nodes.

   d. Update `ssh` keys for new service nodes.

      1) Create a backup copy of your `boot:/root/.ssh/known_hosts` file.

      2) Delete your `boot:/root/.ssh/known_hosts` file.

      3) Run the `/var/opt/cray/install/shell_ssh.sh` script, which creates a new `known_hosts` file. These keys are used for `ssh` commands to the blades, including `pdsh` commands that are called by `xtshutdown` and `/etc/init.d/lustre` and possibly by actions specified in the boot automation file on the SMW.

         ```
         boot:~ # /var/opt/cray/install/shell_ssh.sh
         ```

   e. Configure the new nodes to support the role that they were added for.

   f. Update SMW boot automation files if new service nodes have been added or removed that are providing services (such as ALPS on login nodes) that are explicitly started by `hostname` in the boot automation file.

   g. Complete booting the system (or reboot using the boot automation file).

6.30.2 Updating the System Configuration While the System Is Booted

To change the system configuration physically while the system is booted, use the `xtwarmswap` command to remove or add one or more blades or to remove or add a high-speed network (HSN) cable.

Caution: When reserving nodes for maintenance, an `admindown` of any node in use by a current batch job can cause a subsequent `aprun` in the job to fail. Instead, it is recommended that a batch subsystem be used to first reserve nodes for maintenance, and then verify that a node is not in use by a batch job prior to setting a node to `admindown`. Contact your Cray service representative to reserve nodes for maintenance.

The `xtwarmswap` command runs on the SMW and coordinates with the `xtnlrd` daemon to take the necessary steps to perform warm swap operations. See the `xtwarmswap`(8) man page for additional information about using the `xtwarmswap` command.

6.30.2.1 Reusing One or More Previously-failed HSN Links

To integrate failed links back into the HSN configuration, the `xtwarmswap` command may be invoked with one of the following:

- `-s` *LCB*, ..., specifying the list of LCBs to bring back up
- `-s all`, to bring in all available LCBs
- `-s none`, to cause a reroute without changing the LCBs that are in use

Procedure 52. Reroute the HSN to use previously-failed links

1. Execute an `xtwarmswap -s` *LCB_names* `-p` *partition_name* to tell the system to reroute the HSN using the specified set of LCBs in addition to those that are currently in use.

   Note: Doing so will clear the alert flags on the specified LCBs automatically. If the warm swap fails, the alert flag will be restored to the specified LCBs.

2. Execute an `xtwarmswap -s all -p` *partition_name* command to tell the system to reroute the HSN using all available links.

The `xtwarmswap` command results in `xtnlrd` performing the same link recovery steps as for a failed link, but with two differences: no alert flags are set, and an `init_new_links` and a `reset_new_links` step are performed to initialize both ends of any links to be used, before new routes are asserted into the Aries routing tables.

The elapsed time for the warm swap synchronization operation is typically about 30 seconds.

6.30.2.2 Reusing One or More Previously-failed Blades, ANCs, or Cabinets

Failed blades have alert flags set on the ASICs and the LCBs. These alert flags must be cleared before the blades, ANCs, or cabinets can be reused.

Perform an `xtwarmswap --add` operation to bring the blades back into the HSN configuration. Doing so clears any alert flags on the added blades and LCBs relating to those blades, initializes the blades, runs the BIOS on the nodes, and initializes the links to the blades.

Procedure 53. Clear all alerts associated with the failed blades/ANCs/cabinets and bring them back into the HSN configuration

1. Ensure that blades/ANCs/cabinets have power.
2. Ensure that an `xtalive` command to all required blades succeeds.
3. Add the blade(s) to the HSN by executing the `xtwarmswap --add` *blade_ID*`,...` command. Note that this command automatically executes a mini-`xtdiscover` command after the warm swap steps have completed successfully. No manual invocation of `xtdiscover`, which gets the new hardware attributes from the added blades, is necessary.

   Because the `xtwarmswap --add` command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.
4. Boot the nodes on the blade(s) by executing the `xtcli boot CNL0` *blade_ID*`,...` command on the SMW.

6.30.2.3 Adding or Removing a High-speed Network Cable from Service

To specify a high-speed network cable to add or remove from service, use the `xtwarmswap --add-cable` *cable* command or the `xtwarmswap --remove-cable` *cable* command, respectively.

These options provide the ability to replace a single cable without removing blades or shutting down the system. These options are not supported if more than a single active partition exists in the system. Also, the `-p`|`--partition` option must not be specified when using these options. The routing of the Cray HSN will be updated to route around the removed cable. In addition, the `--linktune` option must not be specified when using the `--remove-cable` option.

6.30.2.4 Planned Removal of a Compute Blade

You can physically remove a compute blade for maintenance or replacement while the system is running; however, the applications using the nodes on the blade to be removed must be allowed to drain, or be killed beforehand.

Procedure 54. Remove a compute blade from service while the system is running

Verify that the proposed system configuration is routable prior to starting this warm swap procedure. Doing this in advance of idling the nodes on the blades to be removed provides assurance that a valid set of nodes is being taken out of service before affecting the system. Log on to the SMW as `crayadm` and execute the command, where `p`*N* is the partition from which the blades are being removed:

```
smw:~> rtr --remove=blade_ID,blade_ID,blade_ID,... \
--cycle-checker --bad-routes pN
```

Caution: This procedure warm swaps a compute blade from service while the system is running. Do not warm swap service blades, unless the blade is an I/O base blade (IBB) that has InfiniBand cards and is an LNET blade. Before attempting to warm swap any service blade, it is advisable to consult with your Cray service representative.

1. Log on to the login node as `root`.
2. Execute the following command to mark the nodes on the compute blade as `admindown`. This tells ALPS not to launch new applications onto them. (This command may also be executed from the boot node as user `root`.)

   ```
   login:~ # xtprocadmin -k s admindown -n blade_ID
   ```

   The arguments to the `-n` option should be the NID values for the nodes on the blade being removed, as shown by executing `xtprocadmin | grep` *bladename*.

   For example, to find the NID values for the nodes on the blade `c0-0c0s2` being removed:

   ```
   login:~ # xtprocadmin | grep c0-0c0s2
        8      0x8  c0-0c0s2n0  compute        up       batch
        9      0x9  c0-0c0s2n1  compute        up       batch
       10      0xa  c0-0c0s2n2  compute        up       batch
       11      0xb  c0-0c0s2n3  compute        up       batch
   ```

3. From the login node, execute the `apstat -n` command to determine if any applications are running on the node you marked `admindown`. In this example, you can see that `apid 675722` is running on all nodes of blade `c0-0c0s2`:

   ```
   login:~ # apstat -n | egrep -w 'NID|8|9|10|11
     8   XT UP  B 32 32  1    4K 16777216 8388608 262144   1 675722
     9   XT UP  B 32 32  1    4K 16777216 8388608 262144   1 675722
    10   XT UP  B 32 32  1    4K 16777216 8388608 262144   1 675722
    11   XT UP  B 32 32  1    4K 16777216 8388608 262144   1 675722
   ```

4. Wait until the applications using the nodes on the blade finish or use the `apkill` *apid* command to kill the application.
5. Log on to the SMW as `crayadm`.

6. Execute the `xtcli halt` *blade_ID* command to halt the blade.

   ```
   smw:~> xtcli halt blade_ID
   ```

7. Execute the `xtwarmswap --remove` *blade_ID* command to remove the compute blade from service. The routing of the Cray HSN will be updated to route around the removed blade.

   The `--remove` stage of the `xtwarmswap` process uses the Aries resiliency infrastructure and takes about 30 seconds to complete.

   ```
   smw:~> xtwarmswap --remove blade_ID
   ```

8. Execute the `xtcli power down` *blade_ID* command, which helps to identify which blade to pull (all lights are off on the blade).

   ```
   smw:~> xtcli power down blade_ID
   ```

9. Physically remove the blade, if desired. To complete this step, see the hardware maintenance and replacement procedures documentation for your Cray system, or contact your Cray Service representative.

Caution: If you cannot reinstall a blade in the empty slot within 2 minutes, install a filler blade assembly in the empty slot; failure to do so can cause other blades in the system to overheat.

6.30.2.5 Planned Installation of a Compute Blade

After a blade has been repaired or when a replacement blade is available, you can use the following procedure to return the blade into service.

Procedure 55. Return a compute blade into service

1. Physically insert the blade into the slot. To complete this step, see the hardware maintenance and replacement procedures documentation for your Cray system, or contact your Cray Service representative.

2. On the SMW, execute the `xtcli power up` *blade_ID* command.

   ```
   smw:~> xtcli power up blade_ID
   ```

3. Ensure that the blade is ready by entering the following command, and wait until the command returns the correct response:

   ```
   smw:~> xtalive blade_ID
   The expected response was received.
   ```

4. Verify the status of the blade controller to ensure that its "Comp state" is "up" and that there are no flags set.

   ```
   smw:~> xtcli status -t bc blade_ID
   ```

5. Bounce the blade.

   ```
   smw:~> xtbounce blade_ID
   ```

6. If the blade or PDC type is different, `su` to `root`, execute the `xtdiscover` command, and then exit `root`. Otherwise, skip this step.

   ```
   smw:~> su - root
   smw:~ # xtdiscover
   smw:~ # exit
   smw:~>
   ```

7. Execute the `xtzap --blade` command to update the BC BIOS, node BIOS, microcontroller, and FPGAs as required.

   ```
   smw:~> xtzap --blade blade_cname
   ```

8. Execute the `xtbounce --linkdown` *blade_ID* command to prepare the blade for the warm swap (takes down all HSN links on the blade).

   ```
   smw:~> xtbounce --linkdown blade_ID
   ```

9. Add the blade(s) to the HSN by executing the `xtwarmswap --add` *blade_ID*`,...` command. This command activates routing on the newly installed blade and automatically executes a mini-`xtdiscover` command once the warm swap steps have completed successfully. No additional manual invocation of `xtdiscover`, which gets the new hardware attributes from the added blades, is necessary.

   ```
   smw:~> xtwarmswap --add blade_ID
   ```

   Because the `xtwarmswap --add` command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.

10. Boot the nodes on the blade(s) by executing the `xtcli boot CNL0` *blade_ID*`,...` command on the SMW.

    ```
    smw:~> xtcli boot CNL0 blade_ID
    ```

11. As `root` on the login node, execute the following command to mark the nodes on the compute blade as `up`. This tells ALPS that new applications may be launched onto those nodes. (This command may also be executed from the boot node as user `root`.)

    ```
    login:~ # xtprocadmin -k s up -n blade_ID
    ```

12. Verify that the blade is up.

    ```
    login:~ # xtprocadmin | grep blade_ID
    ```

6.31 Changing the Location to Log syslog-ng Information

Syslog messages from the service partition are only present on the SMW in `/var/opt/cray/log/`*sessionid*. The log system does not support customization of the configuration files, but if you wish to have custom log formats, you may configure the log system to forward all system logs from the SMW to your site-provided log server. See the `intro_llm`(8) and `CLEinstall.conf`(5) man pages for more information.

6.32 Cray Lightweight Log Management (LLM) System

The Cray Lightweight Log Management (LLM) system is the log infrastructure for Cray systems and must be enabled for systems to successfully log events. At a high level, a library is used to deliver messages to `rsyslog` utilizing the RFC 5424 protocol; `rsyslog` transports those messages to the SMW and places the messages into log files.

For an overview of LLM, see the `intro_LLM`(8) man page.

For an overview of the LLM log files, see the `intro_LLM_logfiles`(5) man page.

The LLM system relies on the *sessionid* that is generated by `xtbootsys`. Therefore, systems must always be booted using `xtbootsys`. If you have multi-part boot procedures or if you use manual procedures, have the process started by an `xtbootsys` session. That session can be effectively empty – it is only needed to initiate a boot *sessionid*. Subsequent `xtbootsys` calls can then use `--session last` or manual processes.

By default, LLM has a log trimming mechanism enabled called `xttrim`. For additional information, see Removing Old Log Files on page 100.

Note: Do not use the `xtgetsyslog` command because it is not compatible with LLM.

6.32.1 Configuring LLM

The LLM system is intended to work as a turnkey system. In most cases little or no configuration is required.

Both the SMW and CLE software installation tools allow certain LLM settings to be defined. The installation tool enforces any of these settings in the LLM configuration file, which means that these settings will not be accidentally lost during an upgrade due to changes in the `llm.conf` file that is distributed with the LLM software.

Note: The `llm.conf` file may be automatically replaced during the upgrade process. The previous version will be renamed `llm.conf.rpmsave`. Settings made in the `SMWinstall.conf` and `CLEinstall.conf` files will be enforced in the new `llm.conf` configuration file. It is important to review the resulting `llm.conf` file after a software update to ensure all settings are as intended.

The most critical parameter is `LLM=` in the `SMWinstall.conf` and `CLEinstall.conf` files. This parameter must be set to `LLM=yes`, which ensures that `enabled=yes` is set in the `llm.conf` file. The `LLM=yes` setting is the only one needed for basic LLM operation.

Important: It is recommended that the minimal possible configuration be performed in order achieve the needed results. Only set or change items if needed. If you must change a setting, change it in the `SMWinstall.conf` and/or `CLEinstall.conf` files if they provide a mechanism to change that value. Only change values in the `llm.conf` configuration file (`/etc/opt/cray/llm/llm.conf`) directly if you require the change **and** the `SMWinstall.conf` and `CLEinstall.conf` files do not provide a mechanism to change that value.

Caution: The `rsyslog.conf` configuration file is not intended to be modified locally. Configurable settings can be found in the `llm.conf` configuration file. Modifications outside of those provided by `llm.conf` may cause failure in other Cray software components, such as `xtdumpsys`. If the provided configuration does not have the needed functionality, it is recommended that the logs be forwarded to another host where custom file processing can be performed without risk to critical software components.

For a description of all LLM configuration settings, see the `llm.conf`(5) man page or the `llm.conf` configuration file.

6.32.2 State Manager LLM Logging

The log data from the State Manager is written to `/var/opt/cray/log/sm-`*yyyymmdd*. The default setting for the State Manager is to enable LLM logging. If LLM or craylog failures occur, State Manager logging is not disrupted. Logging then reverts to behavior that is very similar to legacy State Manager logging, which is also used when State Manager LLM logging is turned off. To disable LLM logging for the State Manager, add the `"-L n"`option to the `/opt/cray/hss/default/bin/rsms` script entry:

```
sm=(/opt/cray/hss/default/bin/state_manager sm "-L n")
```

6.32.3 Boot Manager LLM Logging

The log data from the Boot Manager is written to `/var/opt/cray/log/bm-`*yyyymmdd*. If the `-L` command line option is used with the `bootmanager` command or if LLM is not enabled, Boot Manager reverts to legacy logging, which writes log data to `/var/opt/cray/log/bm.out`. This is a less satisfactory logging method because each Boot Manager restart creates a new log and moves the previous log to `bm.out.1`. A third restart can possibly cause recent log data to be lost.

6.32.4 LLM Configuration Tips

- If your RAID controllers do not have IP addresses that begin with 10.1.0., make sure you specify the `llm_raid_ip=` option in your `SMWinstall.conf` file. Failure to set this correctly will result in the RAID logs not going into their specified location of `/var/opt/cray/log/raid-`*yyyymmdd*.
- Verify that your `/var/opt/cray/log` directory is on a different disk than the root hard drive. The subdirectories of `/var/opt/cray` should be links to a different disk.
- If you delete an active log file while `rsyslog` is running, `rsyslog` will continue to write to the file handle even though there is no longer an entry for the file in the directory table of contents. Once `rsyslog` exits, all references to that file handle are gone, so the contents will be lost. To delete a currently open log file, the suggested approach is to rename or remove the file and then `hup rsyslog` (`/etc/init.d/cray-syslog hup`) to tell it to reopen files.
- Make sure your site log host system can handle the log files load. Otherwise, the messages will back up on the SMW and cause unexpected behavior.

Managing Services [7]

This chapter describes how to manage Cray system services to best use the system or to modify a service.

For a list of administrator accounts that enable you to access these functions, see Administering Accounts on page 117.

7.1 Configuring the SMW to Synchronize to a Site NTP Server

The components of the Cray system synchronize time with the System Management Workstation (SMW) through Network Time Protocol (NTP). By default, the NTP configuration of the SMW is configured to stand alone; however, the SMW can optionally be configured to synchronize with a site NTP server. Use the following procedure to configure the SMW to synchronize to a site NTP server.

Procedure 56. Configuring the SMW to synchronize to a site NTP server

1. Stop the NTP server by issuing the `/etc/init.d/ntp stop` command; this command must be executed as user `root`:

   ```
   smw:~ # /etc/init.d/ntp stop
   ```

2. Edit the `/etc/ntp.conf` file on the SMW to point to the new server.

3. Restart the NTP server by issuing the `/etc/init.d/ntp restart` command:

   ```
   smw:~ # /etc/init.d/ntp start
   ```

The SMW can continue to update the rest of the system by proxy. By default, the SMW qualifies as a stratum 3 (local) NTP server. For more information about NTP, refer to the Linux documentation.

7.2 Synchronizing Time of Day on Compute Node Clocks with the Clock on the Boot Node

A network time protocol (NTP) client, `ntpclient`, is available to install on compute nodes. By default, `ntpclient` is not installed. When installed, the time of day on compute node clocks is synchronized with the clock on the boot node.

Without this feature, compute node clocks will drift apart over time, as much as 18 seconds a day. When `ntpclient` is installed on the compute nodes, the clocks drift apart for a four-hour calibration period and then slowly converge on the time reported by the boot node.

Note: The standard Cray system configuration includes an NTP daemon (`ntpd`) on the boot node that synchronizes with the clock on the SMW. Additionally, the service nodes run `ntpd` to synchronize with the boot node.

To install the `ntpclient` RPM in the compute node boot image, edit the `shell_bootimage_`*LABEL*`.sh` script and specify `CNL_NTPCLIENT=y`, and then update the CNL boot image. Optionally, you can enable this feature as part of a CLE software upgrade by setting `CNL_ntpclient=yes` in the `CLEinstall.conf` file before the `CLEinstall` program is run.

On compute nodes, the computational overhead for `ntpclient` is negligible and a small increase (800K) to the memory footprint will be incurred. Minimal network overhead for the boot node is required to process NTP requests. For each compute node on the system, the boot node will send and receive one packet every 15 minutes. Even on very large Cray systems, the boot node will process fewer than 25 transactions a second to support `ntpclient` requests.

7.3 Adding and Starting a Service Using Standard Linux Mechanisms

Services can be added to the service nodes by using standard Linux mechanisms, such as executing the `chkconfig` command while in the `xtopview` utility on the boot node or executing `/etc/init.d/`*servicename* `start|stop|restart` (which starts, stops, or restarts a service immediately) on the service node. This is the recommended approach for most services.

7.4 Creating a Snapshot of `/var`

The `/var` directory on a Cray system can be configured either as persistent (see *Installing and Configuring Cray Linux Environment (CLE) Software*, S–2444) or nonpersistent. In the latter case, the `/var` directory is volatile, and its initial contents are rebuilt at boot time from a skeleton archive, `/.shared/var-skel.tgz`.

The advantage of using a nonpersistent `/var` directory is ease of management. Each time the system is rebooted, the `/var` directory is freshly recreated from the central skeleton file, so accumulation of files and potential corruption of files with the `/var` directory is much less of a concern. However, because the contents of `/var` are not saved, if there is a need to update the initial contents of the `/var` directory (for example, when a new package requires a directory), the skeleton archive must be updated.

The `xtpkgvar` command creates a compressed tar file with a skeleton snapshot of the `/var` directory. To add files to the directory, make changes in the `xtopview` shell to the `/var` directory and take a snapshot of it with the `xtpkgvar` command.

Caution: Use the `xtpkgvar` command only when you are configuring the shared-root file system. The `xtpkgvar` command is used by the `CLEinstall` utility.

For more information, see the `xtpkgvar`(8) man page.

7.5 Setting Soft and Hard Limits to Prevent Login Node Hangs

A login node can be caused to hang or become nearly unresponsive by having all available processes on the node in use. A hang of this type can be identified primarily by the presence of `cannot fork` error messages, but it is also associated with an unusually large number of processes running concurrently, the machine taking several minutes to make a prompt available, or never making a prompt available. In the case of an overwhelming number of total processes, it is often a large number of the same process overwhelming the system, which indicates a `fork()` system call error in that particular program.

This problem can be prevented by making a few changes to configuration files in `/etc` on the shared root of the login node. These configurations set up the `ulimit` built-in and the Linux Pluggable Authentication Module (PAM) to enforce limits on resources as specified in the configuration files. There are two types of limits that can be specified, a soft limit and a hard limit. Users receive a warning when they reach the soft limit specified for a resource, but they can temporarily increase this limit up to the hard limit using the `ulimit` command. The hard limit can never be exceeded by a normal user. Because of the shared root location of the configuration files, the changes must be made from the boot node using the `xtopview` tool.

Procedure 57. Preventing login node hangs by setting soft and hard limits

1. On the boot node type the following in order to make changes to the shared root, where `login` is the class name for login nodes.

   ```
   boot:~ # xtopview -c login
   ```

2. Next, add the following lines to the `/etc/security/limits.conf` file, where *soft_lim_num* and *hard_lim_num* are the number of processes at which you would like the hard and soft limits enforced. The `*` represents "apply to all users" but can also be configured to apply specific limits by user or group (see the `limits.conf` file's comments for further options).

   ```
   class/login:/ # vi /etc/security/limits.conf
   * soft nproc soft_lim_num
   * hard nproc hard_lim_num
   ```

   Save the file.

3. Verify that the following line is included in the appropriate PAM configuration files for any authentication methods for which you want limits enforced; the PAM configuration files are located in the `/etc/pam.d/` directory. For example, to enforce limits for users connecting through `ssh`, add the `pam_limits.so` line to the file `/etc/pam.d/sshd`. Other applicable authentication methods to include also are `su` in the file `/etc/pam.d/su` and local logins in `/etc/pam.d/login`.

   ```
   session required pam_limits.so
   ```

   For more information about the Pluggable Authentication Module (PAM), see the `PAM`(8) man page.

4. Type `exit` to return to the normal prompt on the boot node; the changes you made should be effective immediately on login nodes.

   ```
   class/login:/ # exit
   boot:~ #
   ```

5. To test that the limits are in place, from a login node type the following command, which should return the number specified as the soft limit for the number of processes available to a user, for example:

   ```
   boot:~ # ssh login
   login:~ # ulimit -u
   ```

   For more information about using the `ulimit` command, see the `ulimit`(P) man page.

7.6 Rack-mount SMW: Create a Cray System Management Workstation (SMW) Bootable Backup Drive

The following procedure creates a bootable backup drive for a rack-mount SMW in order to replace the primary drive if the primary drive fails. When this procedure is completed, the backup drive on the SMW will be a bootable replacement for the primary drive when the backup drive is plugged in as or cabled as the primary drive.

Procedure 58. Rack-mount SMW: Creating an SMW bootable backup drive

Important: The disk device names shown in this procedure are only examples. You should substitute the actual disk device names for your system. The boot disk is `phy7` and is slot 0, and the bootable backup disk is `phy6` and is slot 1.

Caution: Shut down the Cray system before you begin this procedure.

Also be aware that there may be a considerable load on the SMW while creating the SMW bootable backup drive.

1. Log on to the SMW as `crayadm` and `su` to `root`.

```
crayadm@smw:~> su -
Password:
smw:~ # ls -al /dev/disk/by-path
total 0
drwxr-xr-x 2 root root 380 Mar 15 13:21 .
drwxr-xr-x 6 root root 120 Mar 11 18:42 ..
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:00:11.0-scsi-0:0:0:0 -> ../../sr0
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:00:12.2-usb-0:3:1.0-scsi-0:0:0:0 -> ../../sdf
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:00:12.2-usb-0:3:1.1-scsi-0:0:0:0 -> ../../sr1
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:00:12.2-usb-0:3:1.1-scsi-0:0:0:1 -> ../../sdg
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:00:13.2-usb-0:2.1:1.0-scsi-0:0:0:0 -> ../../sde
lrwxrwxrwx 1 root root  10 Mar 11 18:42 pci-0000:00:13.2-usb-0:2.1:1.0-scsi-0:0:0:0-part1 -> ../../sde1
lrwxrwxrwx 1 root root  10 Mar 11 18:42 pci-0000:00:13.2-usb-0:2.1:1.0-scsi-0:0:0:0-part2 -> ../../sde2
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:05:00.0-sas-phy4-...
lrwxrwxrwx 1 root root  10 Mar 14 15:57 pci-0000:05:00.0-sas-phy4-...
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:05:00.0-sas-phy5-...
lrwxrwxrwx 1 root root  10 Mar 14 16:00 pci-0000:05:00.0-sas-phy5-...
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:05:00.0-sas-phy6-...
lrwxrwxrwx 1 root root  10 Mar 15 13:21 pci-0000:05:00.0-sas-phy6-...
lrwxrwxrwx 1 root root  10 Mar 15 13:21 pci-0000:05:00.0-sas-phy6-...
lrwxrwxrwx 1 root root   9 Mar 11 18:42 pci-0000:05:00.0-sas-phy7-...
lrwxrwxrwx 1 root root  10 Mar 11 18:42 pci-0000:05:00.0-sas-phy7-...
lrwxrwxrwx 1 root root  10 Mar 11 18:42 pci-0000:05:00.0-sas-phy7-...
```

2. Standardize the SMW's boot-time drive names with the Linux run-time drive names.

   Important: If the SMW configuration files on the SMW root drive have been modified already (because your site has completed this step at least once after installing your updated SMW base operating system), skip to step 3; otherwise, complete this step to standardize the SMW's boot-time drive names with the Linux run-time drive names.

   Set up ordered drives on your rack-mount SMW.

   a. Identify the installed SMW drive model numbers, serial numbers, and associated Linux device (`/dev`) names.

Execute the `smwmapdrives.sh` script on the SMW to identify local (internal) drives mounted in the SMW and provide their Linux device (`/dev`) names.

Note: Effective with the SMW 7.2.UP00 release, the `smwmapdrives.sh` script is provided both as a separate file in the release and in the base operating system RPM. Prior to doing this update you will need to use the separate file, but when backing up your SMW at a later date you can use the installed version.

```
smw:~ # ./smwmapdrives.sh
     List of SMW-installed disk drives
     ---------------------------------
     Physical slot 0:
                /dev/sda
                /dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS
                /dev/disk/by-id/scsi-SATA_FUJITSU_MHZ2160_K85DTB227RDS
                /dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0
           Physical slot 1:
                /dev/sdc
                /dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RD7
                /dev/disk/by-id/scsi-SATA_FUJITSU_MHZ2160_K85DTB227RD7
                /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0
           Physical slot 2:
                /dev/sdd
                /dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RF3
                /dev/disk/by-id/scsi-SATA_FUJITSU_MHZ2160_K85DTB227RF3
                /dev/disk/by-path/pci-0000:05:00.0-sas-phy5-0x4433221105000000-lun-0
           Physical slot 3:
                /dev/sdb
                /dev/disk/by-id/ata-ST9500620NS_9XF0665V
                /dev/disk/by-id/scsi-SATA_ST9500620NS_9XF0665V
                /dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0
           Physical slot 4:
                NOT INSTALLED
           Physical slot 5:
                NOT INSTALLED
```

The device names for `by-id` are persistent and will reference the drive, regardless of the slot in which the drive is installed.

`by-path` names reference a physical drive slot only and do not identify the drive installed in that slot. This is the naming used by default for the logging and database drives when the SMW was installed. This `by-path` name is used to specifically install logging and database file systems because the `by-id` device names refer to the physical drive slots expected to be used for those file systems and are provided as the default examples in the SMW installation configuration process.

The `/dev/sd`*X* drive names are not persistent, except for the `/dev/sda` drive name, which always refers to the drive installed in physical slot `0` (zero) of the SMW. The other `/dev/sd`*X* names can change with each SMW boot

and will change if drives are added, removed, or reordered in the SMW slots. For this reason, the `/dev/sda` drive name can only be used for the rack-mount SMW.

Choose either the `by-id` naming or the `by-path` naming as the site administrative policy for managing the SMW-install disk drives. The following documentation provides the steps necessary to implement this selection on the SMW prior to creating an SMW bootable backup drive.

b. Back up the following files before proceeding:

```
smw# cp -p /boot/grub/device.map /boot/grub/device.map-YYYYMMDD
smw# cp -p /boot/grub/menu.lst /boot/grub/menu.lst-YYYYMMDD
smw# cp -p /etc/fstab /etc/fstab-YYYYMMDD
```

c. Edit the `grub device.map` file to reflect physical drive locations.

To provide a direct mapping of the SMW disk drive physical slots to the boot loader (BIOS and `grub`) drive names, the `device.map` mapping file used by `grub` should be replaced. Perform the following steps to install new `device.map` file entries to effect this mapping.

1) Edit the `grub device.map` file.

2) Delete all lines.

3) Enter the following lines into the file. These lines show each drive slot's physical location mapped to its boot-time `hd?` name.

Note: `by-id` names should not be used in the `device.map` file.

```
# Dell Rackmount r805 SMW
# grub(8) device mapping for boot-drive identification
# hd? numbers are being mapped to their physical
(hd0)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0
(hd1)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0
(hd2)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy5-0x4433221105000000-lun-0
(hd3)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0
(hd4)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy3-0x4433221103000000-lun-0
(hd5)   /dev/disk/by-path/pci-0000:05:00.0-sas-phy2-0x4433221102000000-lun-0
```

d. Modify the SMW boot drive `/etc/fstab` file to utilize `by-id` or `by-path` naming.

Modify the SMW file system mounting configuration file to utilize SMW disk `by-id` or `by-path` naming. Complete this step to replace any `/dev/sd`*X* disk partition references.

Note: Use the output of the `smwmapdrives.sh` script in step 2.a as a reference for drive names.

Edit `/etc/fstab`, replacing drive `/dev/sd`*X* references with either the `by-id` or `by-path` name's corresponding device name.

When a reference to `/dev/sda1` is being replaced, replace it with the

corresponding "partition" file system suffixed with `-part1`. File system partitions for `/dev/sda` are indicated by the numeral appended to the device name; for example, `/dev/sda1` refers to partition `1` on `/dev/sda`. Replace it with the `by-id` and `by-path` device names and suffix the device name with `-part1`, keeping the same numeral identification.

For example, if the `root` and `swap` file systems are currently configured to mount `/dev/sda2`, they should be changed. Using the `by-id` device name from the example in step 2.a, the `fstab` lines would change from:

```
/dev/sda1 swap              swap       defaults              0 0
/dev/sda2 /                 ext3       acl,user_xattr        1 1
```

to:

```
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS-part1  swap  swap   defaults        0 0
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS-part2  /     ext3   acl,user_xattr  1 1
```

e. Modify `/boot/grub/menu.lst` to reflect the `device.map` BIOS/boot-up drive changes for the `sd`*X* remapping.

   The same device name replacement performed on `/etc/fstab` should also be performed on the `grub` bootloader `/boot/grub/menu.lst` configuration file. All references to `/dev/sd`*X* devices should be replaced with corresponding `by-id` or `by-path` device names.

f. Invoke the `grub` utility to reinstall the SMW boot loader on the primary boot drive.

   Once the changes to `device.map`, `fstab`, and `menu.lst` have been completed, the `grub` bootloader boot blocks must be updated to reflect changes to the device names. Complete this step to update the boot loader on the boot drive.

   Invoke the `grub` utility and reinstall SMW root-drive boot blocks.

```
smw:~ # grub --no-curses
    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)
 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename. ]
grub> root (hd0,1)
  root (hd0,1)
    Filesystem type is ext2fs, partition type 0x83
grub> setup (hd0)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/e2fs_stage1_5" exists... yes
 Running "embed /boot/grub/e2fs_stage1_5 (hd0)"... 17 sectors \
are embedded. Succeeded
 Running "install /boot/grub/stage1 (hd0)  (hd0)1+17 p (hd0,1)/boot/grub/stage2 \
/boot/grub/menu.lst"... succeeded
       Done.
       grub> quit
```

3. If the backup drive disk partition table already exists and the partition table on the backup drive matches the partition table that is on the primary boot drive, skip this step; otherwise, create the backup drive disk partition table.

   In this example, the partition table consists of the following:

   - Slice 1: 4 GB Linux swap partition
   - Slice 2: Balance of disk space used for the root file system

   a. Use the `fdisk` command to display the boot disk partition layout. (Example output spacing was modified to fit on the printed page.)

```
smw:~ # fdisk -lu /dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0
Disk /dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0: 250.0 GB, \
268435456000 bytes
255 heads, 63 sectors/track, 19457 cylinders, total 312581808 sectors
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x00000082

                    Device
/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part1 \
                                  Boot   Start       End     Blocks  Id  System
                                            63  16771859   8385898+  82  Linux swap / Solaris
/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part2 \
                                  Boot   Start       End     Blocks  Id  System
                                    * 16771860 312576704 147902422+  83  Linux
```

   b. Use the `fdisk` command to configure the bootable backup disk partition layout. Set the bootable backup disk partition layout to match the boot disk partition layout. First, clear all of the old partitions using the `d` command within `fdisk`; next create a Linux swap and a Linux partition; and then write your changes to the disk. For help, type **`m`** within `fdisk` (see the following sample output, spacing was modified to fit on the printed page.)

```
smw:~ # fdisk -u /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0

The number of cylinders for this disk is set to 19457.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0: 250.0 GB, \
268435456000 bytes
255 heads, 63 sectors/track, 19457 cylinders, total 312581808 sectors
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x00000080

                    Device
/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part1 \
                                  Boot   Start       End     Blocks  Id  System
                                            63  16771859      83828  82  Linux swap / Solaris
Partition 1 does not end on cylinder boundary.
/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2 \
                                  Boot   Start       End     Blocks  Id  System
                                       167719 312581807 156207044+  83  Linux
```

```
Command (m for help): d
Partition number (1-4): 2

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First sector (63-312581807, default 63): (Press the Enter key)
Using default value 63
Last sector, +sectors or +size{K,M,G} (63-312581807, default 312581807): 16771859
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 82
Changed system type of partition 1 to 82 (Linux swap / Solaris)

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First sector (16771860-312581807, default 16771860): (Press the Enter key)
Using default value 16771860
Last sector, +sectors or +size{K,M,G} (16771860-312581807, default 312581807): (Press the Enter key)
Using default value 312581807

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

c. Display the boot backup disk partition layout and confirm it matches your phy7 sector information.

```
smw:~ # fdisk -lu /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0
Disk /dev/disk/by-path/pci-0000:05:00.0-sas-phy6:1-0x4433221106000000:0-lun0: 250.0 GB, \
268435456000 bytes
255 heads, 63 sectors/track, 19457 cylinders, total 312581808 sectors
```

4. Initialize the swap device.

```
smw:~ # mkswap /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part1
mkswap: /dev/disk/by-path/pci-0000:05:00.0-sas-phy6:1-0x4433221106000000:0-lun0-part1:
warning: don't erase bootbits sectors
        (DOS partition table detected). Use -f to force.
Setting up swapspace version 1, size = 8385892 KiB
no label, UUID=c0ef22ac-b405-4236-855b-e4a09b6e94ed
```

5. Create a new file system on the backup drive `root` partition by executing the `mkfs` command.

```
smw:~ # mkfs -t ext3 \
/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
9248768 inodes, 36976243 blocks
1848812 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
1129 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872

Writing inode tables:   done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

6. Mount the new backup `root` file system on `/mnt`.

```
smw:~ # mount /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2 \
/mnt
```

7. Confirm that the backup `root` file system is mounted.

```
smw:~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2            303528624   6438700 281671544   3% /
udev                   1030332       116   1030216   1% /dev
/dev/sdb2            306128812    195568 290505224   1% /mnt
```

The running `root` file system device is the one mounted on `/`.

8. Dump the running `root` file system to the backup drive.

```
smw:~ # cd /mnt
smw:~ # dump 0f - /dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part2 \
| restore rf -
  DUMP: WARNING: no file `/etc/dumpdates'
  DUMP: Date of this level 0 dump: Tue Mar 15 13:43:17 2011
  DUMP: Dumping /dev/sda2 (/) to standard output
  DUMP: Label: none
  DUMP: Writing 10 Kilobyte records
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 7898711 blocks.
  DUMP: Volume 1 started with block 1 at: Tue Mar 15 13:44:40 2011
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
restore: ./lost+found: File exists
  DUMP: 79.34% done at 20890 kB/s, finished in 0:01
  DUMP: Volume 1 completed at: Tue Mar 15 13:52:13 2011
  DUMP: Volume 1 7908080 blocks (7722.73MB)
  DUMP: Volume 1 took 0:07:33
  DUMP: Volume 1 transfer rate: 17457 kB/s
  DUMP: 7908080 blocks (7722.73MB)
  DUMP: finished in 453 seconds, throughput 17457 kBytes/sec
  DUMP: Date of this level 0 dump: Tue Mar 15 13:43:17 2011
  DUMP: Date this dump completed:  Tue Mar 15 13:52:13 2011
  DUMP: Average transfer rate: 17457 kB/s
  DUMP: DUMP IS DONE
```

9. Modify the backup drive's `fstab` and `menu.lst` files to reflect the backup drive's device, replacing the primary drive's device name.

   Note: This step is necessary only if `by-id` names are used. If `by-path` names are being utilized for the `root` and `swap` devices, changes are not necessary; these devices reference physical slots, and the backup drive will be moved to the same physical slot (slot `0`) when replacing a failed primary boot drive.

   a. Edit `/mnt/etc/fstab`. Replace the `root` and `swap` partitions' `by-id` device names with those used for this backup device, replacing the original disk device name.

      For example, change:

```
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS-part1  swap  swap  defaults
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS-part2  / ext3 acl,user_xattr
```

      to:

```
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RD7-part1  swap  swap  defaults
/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RD7-part2  / ext3 acl,user_xattr
```

b. Edit `/mnt/boot/grub/menu.lst`. Replace the `root=` and `resume=` device names with those used for this backup device, replacing the original disk device name.

Note: The `root=` entry normally refers to partition `-part2`, and the `resume=` entry normally refers to partition `-part1`; these partition references **must** be maintained.

For example, replace the `menu.lst` configuration references of:

```
root=/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RDS-part2
```

with:

```
root=/dev/disk/by-id/ata-FUJITSU_MHZ2160BK_G2_K85DTB227RD7-part2
```

or similarly with the `by-id` device names, if those are preferred.

Replace the `resume=` references similarly.

10. Install the `grub` boot loader.

To make the backup drive bootable, reinstall the `grub` boot facility on that drive.

Caution: Although all of the disks connected to the SMW are available to the system, `grub` only detects the first 16 devices. Therefore, if you add a disk to the SMW **after** the SMW is connected to the boot RAID, it is advisable to reboot the SMW **before** continuing this procedure.

a. Create a unique file on the backup drive to be used to identify that drive to `grub` boot facility.

```
smw:~ # cd /
smw:~ # touch /mnt/THIS_IS_6
```

b. Invoke the `grub` boot utility. Within the `grub` boot utility:

1) Execute the `find` command to locate the drive designation that `grub` uses.

2) Select the drive to which the boot blocks will be installed with the `root` command.

3) Use the `setup` command to set up and install the `grub` boot blocks on that drive.

Note: The Linux `grub` utility and boot system **always** refer to drives as `hd`, regardless of the actual type of drives.

For example:

```
smw:~ # grub --no-curses
    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)
 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename. ]
grub> find /THIS_IS_6
(hd2,1)
grub> root (hd2,1)
root (hd2,1)
 Filesystem type is ext2fs, partition type 0x83
grub> setup (hd2)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/e2fs_stage1_5" exists... yes
 Running "embed /boot/grub/e2fs_stage1_5 (hd2)"...  17 sectors are embedded.
succeeded
 Running "install /boot/grub/stage1 (hd2) (hd2)1+17 p (hd2,1)/boot/grub/stage2 \
/boot/grub/menu.lst"... succeeded
Done.
grub> quit
```

Important: For rack-mount SMWs, `grub` recreates `device.map` with the short names, not the persistent names. The `/dev/sd`*x* names must not be trusted. Always use `find` the next time you execute `grub` because it is possible that `grub root` may **not** be `hd2` the next time you execute `grub`.

11. Unmount the backup root partition.

    ```
    smw:~ # umount /mnt
    ```

    The drive is now bootable once plugged in or cabled as the primary drive.

7.7 Desk-side SMW: Create a System Management Workstation (SMW) Bootable Backup Drive

The following procedure creates a System Management Workstation (SMW) bootable backup drive for a desk-side SMW in order to replace the primary drive if the primary drive fails. When this procedure is completed, the backup drive on the SMW will be a bootable replacement for the primary drive when the backup drive is plugged in as or cabled as the primary drive.

Note: In the following procedure, `/dev/sd`*X*`2` is the SMW disk (either `/dev/sdb2` or `/dev/sdc2`).

Procedure 59. Desk-side SMW: Creating an SMW bootable backup drive

Important: The disk device names shown in this procedure are only examples. You should substitute the actual disk device names for your system. For example, on an SMW with three SMW disks, the boot disk is `/dev/sda` and the bootable backup disk is `/dev/sdc`; on an SMW with two SMW disks, the boot disk is `/dev/sda` and the bootable backup disk is `/dev/sdb`.

Caution: Shut down the Cray system before you begin this procedure.

Also be aware that there may be a considerable load on the SMW while creating the SMW bootable backup drive.

1. Log on to the SMW as `crayadm` and `su` to `root`.

```
crayadm@smw:~> su - root
smw:~ #
```

2. If the backup drive disk partition table already exists and the partition table on the backup drive matches the partition table that is on the primary boot drive, skip this step; otherwise, create the backup drive disk partition table.

   Note: For optimal performance, the source and destination disks should be on different buses; drive slots 0 and 1 are on a different bus than drive slots 2 and 3.

   In this example, the partition table consists of the following:

   - Slice 1: 4 GB Linux swap partition
   - Slice 2: Balance of disk space used for the root file system

   a. Use the `fdisk` command to display the boot disk partition layout.

```
smw:~ # fdisk -lu /dev/sda
Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors
Units = sectors of 1 * 512 = 512 bytes

  Device   Boot     Start         End      Blocks   Id  System
/dev/sda1              63     8401994     4200966   82  Linux swap / Solaris
/dev/sda2   *     8401995   625137344   308367675   83  Linux
```

   b. Use the `fdisk` command to configure the bootable backup disk partition layout. Set the bootable backup disk partition layout to match the boot disk partition layout. First, clear all of the old partitions using the `d` command within `fdisk`; next create a Linux swap and a Linux partition; and then write your changes to the disk. For help, type **`m`** within `fdisk` (see the following sample output).

```
smw:~ # fdisk -u /dev/sdb

The number of cylinders for this disk is set to 38913.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
```

```
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK).

Command (m for help): p
Disk /dev/sdb: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1              63     8401994     4200966   82  Linux swap
/dev/sdb2         8401995   625105214   308351610   83  Linux

Command (m for help): d
Partition number (1-5): 2
Command (m for help): d
Selected partition 1
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1
First sector (63-625105215, default 63): (Press the Enter key)
Using default value 63
Last sector or +size or +sizeM or +sizeK (63-625105215, default 625105215): 8401994

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 82
Changed system type of partition 1 to 82 (Linux swap / Solaris)

Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 2
First sector (8401995-625105215, default 8401995): (Press the Enter key)
Using default value 8401995
Last sector or +size or +sizeM or +sizeK (8401995-625105215, default 625105215): \
  (Press the Enter key)
Using default value 625105215

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

c. Display the boot backup disk partition layout.

```
smw:~ # fdisk -lu /dev/sdb
Disk /dev/sdb: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors
Units = sectors of 1 * 512 = 512 bytes


  Device   Boot     Start        End       Blocks   Id  System

/dev/sdc1              63     8401994     4200966   82  Linux swap / Solaris
/dev/sdc2   *     8401995   625137344   308367675   83  Linux
```

3. Initialize the swap device.

   ```
   smw:~ # mkswap /dev/sdb1
   ```

4. Standardize the `/etc/fstab` and `grub` disk device names.

   The device names that the installation process writes into the `/boot/grub/menu.lst` file are UDEV-based names (for example, `/dev/disk/`*by-id*`/scsi-SATA_ST3320620AS_922J3-part2` or `/dev/disk/`*by-id*`/ata-ST3320620A_9QFA85PV-part2`) instead of the more commonly used device names (for example, `/dev/sda2` or `/dev/hda2`). In the following procedures, edit the `/boot/grub/menu.lst` file to change **only** the long UDEV-based name to the shorter, commonly used device name reflected in the output of the `df` command on your system.

   If the device names have already been standardized, skip to step 5.

   Caution: Mistakes in the `/boot/grub/menu.lst` file will affect your ability to boot the SMW.

   a. SLES 11 sets up `/etc/fstab` and `/boot/grub/menu.lst` with UDEV-based names for the root device. For example:

```
smw:~ # head -2 /etc/fstab
/dev/disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part2 /     ext3  acl,user_xattr  1 1
/dev/disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part1 swap  swap  defaults        0 0

smw:~ # more /boot/grub/menu.lst
###Don't change this comment - YaST2 identifier: Original name: linux###
title SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/disk/by-id/ata-ST3320620AS_5QF00F84-part2 \
    resume=/dev/sda1 splash=silent crashkernel=256M-:128M@16M
    showopts vga=0x31a initrd /boot/initrd-2.6.27.19-5-default

###Don't change this comment - YaST2 identifier: Original name: failsafe###
title Failsafe -- SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/disk/by-id/ata-ST3320620AS_5QF00F84-part2 showopts \
    ide=nodma apm=off noresume edd=off powersaved=off nohz=off highres=off
    processor.max_cstate=1 x11failsafe vga=0x31a
    initrd /boot/initrd-2.6.27.19-5-default
```

b. Execute the `df` command to get the name of the device to use in the `/etc/fstab` and `/boot/grub/menu.lst` files to replace the long UDEV-based device name. Then edit your `/etc/fstab` and `/boot/grub/menu.lst` files appropriately.

1) Execute the `df` command to get the name of the device to use in the `/etc/fstab` and `/boot/grub/menu.lst` files to replace the long UDEV-based device name. For example:

```
smw:# df
    Filesystem   1K-blocks      Used Available Use% Mounted on
    /dev/sda2    303528624  40652904 247457340  15% /
    udev           1030780       460   1030320   1% /dev
```

2) Save a copy of your `/etc/fstab` and `/boot/grub/menu.lst` files.

```
smw:# cp -p /etc/fstab /etc/fstab.save
smw:# cp -p /boot/grub/menu.lst /boot/grub/menu.lst.save
```

3) Edit your `/etc/fstab` file appropriately; you will use the device name (dev) you got from the `df` command output. In this example, the "1" and "2" refer to the partition names on the device. Change the following lines, which changes the long name `disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part2` to `sda2` and changes `disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part1` to `sda1`. Ensure that your swap is on `sda1`:

```
/dev/disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part2 /     ext3  acl,user_xattr  1 1
/dev/disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part1 swap  swap  defaults        0 0
```

to:

```
/dev/sda2   /      ext3   acl,user_xattr   1 1
/dev/sda1   swap   swap   defaults         0 0
```

4) Edit your `/boot/grub/menu.lst` file appropriately; use the device name (dev) you got from the `df` command output. Change the long name `disk/by-id/scsi-SATA_ST3320620AS_9QF922J3-part2` to `sda2`. Change the following lines:

```
title SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/disk/by-id/ata-ST3320620AS_5QF00F84-part2 \
    resume=/dev/sda1 splash=silent crashkernel=256M-:128M@16M showopts vga=0x31a
```

to:

```
title SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/sda2 resume=/dev/sda1 splash=silent \
    crashkernel=256M-:128M@16M showopts vga=0x31a
```

and change the following lines:

```
title Failsafe -- SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/disk/by-id/ata-ST3320620AS_5QF00F84part2 \
    showopts ide=nodma apm=off noresume edd=off powersaved=off nohz=off \
    highres=off processor.max_cstate=1 x11failsafe vga=0x31a
```

to:

```
title Failsafe -- SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/sda2 showopts ide=nodma apm=off noresume edd=off \
    powersaved=off nohz=off highres=off processor.max_cstate=1 x11failsafe vga=0x31a
```

5) Verify that the edited files are correct and matches the output of the `df` command.

```
smw:~ # head -2 /etc/fstab
/dev/sda2   /     ext3   acl,user_xattr   1 1
/dev/sda1   swap  swap   defaults         0 0
```

```
smw:~ # more /boot/grub/menu.lst
###Don't change this comment - YaST2 identifier: Original name: linux###
title SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default root=/dev/sda2 \
    resume=/dev/sda1 splash=silent crashkernel=256M-:128M@16M showopts
    initrd /boot/initrd-2.6.27.19-5-default

###Don't change this comment - YaST2 identifier: Original name: failsafe###
title Failsafe -- SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default \
    root=/dev/sda2 showopts ide=nodma apm=off noresume edd=off
    powersaved=off nohz=off highres=off initrd /boot/initrd-2.6.27.19-5-default
```

5. Update the `grub` device table to recognize any new drives added since the initial operating system installation.

Caution: Although all of the disks connected to the SMW are available to the system, `grub` only detects the first 16 devices. Therefore, if you add a disk to the SMW **after** the SMW is connected to the boot RAID, it is advisable to reboot the SMW **before** continuing this procedure.

a. Back up the current `grub` device mapping file.

```
smw:~ # mv /boot/grub/device.map /boot/grub/device.map-YYYYMMDD
```

b. Invoke the `grub` utility to create a new device mapping file.

```
smw:~ # grub --device-map=/boot/grub/device.map
Probing devices to guess BIOS drives. This may take a long time.
  GNU GRUB  version 0.97  (640K lower / 3072K upper memory)
grub> quit
```

The file `/boot/grub/device.map` is now updated to reflect all drives, utilizing the standardized drive naming. This file can be viewed for verification; for example:

```
smw:~ # cat /boot/grub/device.map
(fd0)   /dev/fd0
(hd0)   /dev/sda
(hd1)   /dev/sdc
```

6. Create a new file system on the backup drive root partition by executing the `mkfs` command.

```
smw:~ # mkfs -t ext3 /dev/sdb2
mke2fs 1.41.1 (01-Sep-2008)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
19275776 inodes, 77091918 blocks
3854595 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
2353 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872, 71663616

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
smw:~ #
```

7. Mount the new backup root file system on `/mnt`.

```
smw:~ # mount /dev/sdb2 /mnt
```

8. Confirm the running root file system device.

```
smw:~ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2            303528624   6438700 281671544   3% /
udev                   1030332       116   1030216   1% /dev
/dev/sdb2            306128812    195568 290505224   1% /mnt
```

The running root file system device is the one mounted on `/`.

9. Dump the running root file system to the backup drive.

```
smw:~ # cd /mnt
smw:~ # dump 0f - /dev/sda2 | restore rf -
DUMP: WARNING: no file `/etc/dumpdates'
DUMP: Date of this level 0 dump: Thu Nov 11 06:55:29 2010
DUMP: Dumping /dev/sda2 (/) to standard output
DUMP: Label: none
DUMP: Writing 10 Kilobyte records
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 4003398 blocks.
DUMP: Volume 1 started with block 1 at: Thu Nov 11 06:57:38 2010
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
restore: ./lost+found: File exists
DUMP: 81.99% done at 10941 kB/s, finished in 0:01
DUMP: Volume 1 completed at: Thu Nov 11 07:04:01 2010
DUMP: Volume 1 4008910 blocks (3914.95MB)
DUMP: Volume 1 took 0:06:23
DUMP: Volume 1 transfer rate: 10467 kB/s
DUMP: 4008910 blocks (3914.95MB)
DUMP: finished in 383 seconds, throughput 10467 kBytes/sec
DUMP: Date of this level 0 dump: Thu Nov 11 06:55:29 2010
DUMP: Date this dump completed:  Thu Nov 11 07:04:01 2010
DUMP: Average transfer rate: 10467 kB/s
DUMP: DUMP IS DONE
```

10. Install the `grub` boot loader.

    To make the backup drive bootable, reinstall the `grub` boot facility on that drive.

Caution: Although all of the disks connected to the SMW are available to the system, `grub` only detects the first 16 devices. Therefore, if you add a disk to the SMW **after** the SMW is connected to the boot RAID, it is advisable to reboot the SMW **before** continuing this procedure.

a. Create a unique file on the backup drive to be used to identify that drive to `grub` boot facility.

```
smw:~ # cd /
smw:~ # touch /mnt/THIS_IS_SDX
```

b. Invoke the `grub` boot utility. Within the `grub` boot utility:

   1) Execute the `find` command to locate the drive designation that `grub` uses.

   2) Select the drive to which the boot blocks will be installed with the `root` command.

   3) Use the `setup` command to set up and install the `grub` boot blocks on that drive.

   Note: The Linux `grub` utility and boot system **always** refer to drives as `hd`, regardless of the actual type of drives.

For example:

```
smw:~ # grub --no-curses
GNU GRUB  version 0.97  (640K lower / 3072K upper memory)
[ Minimal BASH-like line editing is supported.  For the first word, TAB^[
lists possible command completions.  Anywhere else TAB lists the possible
completions of a device/filename. ]
grub> find /THIS_IS_SDX
find /THIS_IS_SDX
(hd1,1)
grub> root (hd1,1)
root (hd1,1)
Filesystem type is ext2fs, partition type 0x83
grub> setup (hd1)
setup (hd1)
Checking if "/boot/grub/stage1" exists... yes
Checking if "/boot/grub/stage2" exists... yes
Checking if "/boot/grub/e2fs_stage1_5" exists... yes
Running "embed /boot/grub/e2fs_stage1_5 (hd1)"...  17 sectors are embedded.
succeeded
Running "install /boot/grub/stage1 (hd1) (hd1)1+17 p
(hd1,1)/boot/grub/stage2 /boot/grub/menu.lst"... succeeded
Done.
grub> quit
```

11. Unmount the backup root partition.

    ```
    smw:~ # umount /dev/sdb2
    ```

    The drive is now bootable once plugged in or cabled as the primary drive.

7.8 Rack-mount SMW: Set Up the Bootable Backup Drive as an Alternate Boot Device

Warning: You **must** be running the SUSE Linux Enterprise Server version 11 Service Pack 3 (SLES 11 SP3) SMW base operating system and a release of SMW 7.1 or later on your SMW in order to perform the procedures in this chapter.

The following procedure modifies a bootable backup drive for a rack-mount SMW in order to boot from and run the rack-mount SMW from the backup root partition.

Important: To boot from this backup drive, the primary boot drive must still be operable and able to boot the `grub` boot blocks installed. If the backup drive is modified to boot as an alternate boot device, it will no longer function as a bootable backup if the primary drive fails.

Procedure 60. Rack-mount SMW: Set up the bootable backup drive as an alternate boot device

Note: This procedure will **not** provide a usable backup drive that can be booted in the event of a primary drive failure.

Caution: The disk device names shown in this procedure are only examples. You should substitute the actual disk device names for your system. The boot disk is `phy7` and is slot `0`, and the bootable backup disk is `phy6` and is slot `1`.

1. Mount the backup drive's root partition.

```
smw:~ # mount /dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2 /mnt
```

2. Create a new boot entry in the `/boot/grub/menu.lst` file. This entry should be a duplicate of the primary boot entry with the following changes:
   - Modify the title to uniquely identify the backup boot entry.
   - Modify the `root (hd0,1)` directive to reflect the `grub` name of the backup drive.
   - Modify the `root=` and `resume=` specifications to reference the backup drive device.

An example `/boot/grub/menu.lst` file follows. Note the new entry for the backup drive. This example references `phy7` (slot `0`) and as the primary drive and `phy6` (slot `1`) as the backup drive.

```
smw:~ # cp -p /boot/grub/menu.lst /boot/grub/menu.lst.20110317
smw:~ # vi /boot/grub/menu.lst
smw:~ # cat /boot/grub/menu.lst
# Modified by YaST2. Last modification on Wed Jun 27 12:32:43 CDT 2012
default 0
timeout 8
##YaST - generic_mbr
gfxmenu (hd0,1)/boot/message
##YaST - activate

###Don't change this comment - YaST2 identifier: Original name: linux###
title SUSE Linux Enterprise Server 11 SP3 - 3.0.26-0.7
    root (hd0,1)
    kernel /boot/vmlinuz-3.0.26-0.7-default \
    root=/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part2 \
    resume=/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part1 \
    splash=silent crashkernel=256M-:128M  showopts vga=0x31a
    initrd /boot/initrd-3.0.26-0.7-default

### New entry allowing a boot of the back-up drive when the primary drive
### is still present.
title BACK-UP DRIVE - SUSE Linux Enterprise Server 11 SP3 - 3.0.26-0.7
    root (hd1,1)
    kernel /boot/vmlinuz-3.0.26-0.7-default \
    root=/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2 \
    resume=/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part1 \
    splash=silent crashkernel=256M-:128M  showopts vga=0x31a
    initrd (hd0,1)/boot/initrd-3.0.26-0.7-default


###Don't change this comment - YaST2 identifier: Original name: failsafe###
title Failsafe -- SUSE Linux Enterprise Server 11 SP3 - 3.0.26-0.7
    root (hd0,1)
    kernel /boot/vmlinuz-3.0.26-0.7-default \
    root=/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part2 \
    showopts ide=nodma apm=off noresume edd=off powersaved=off \
    nohz=off highres=off processor.max_cstate=1 nomodeset x11failsafe vga=0x31a
    initrd /boot/initrd-3.0.26-0.7-default
```

3. Modify the backup drive's `/etc/fstab` file to reference the secondary drive slot rather than the first drive slot.

   a. Examine the backup drive's `fstab` file.

```
smw:~ # cp -p /mnt/etc/fstab /mnt/etc/fstab.20110317
smw:~ # cat /mnt/etc/fstab
/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part1 \
swap    swap   defaults        0 0
/dev/disk/by-path/pci-0000:05:00.0-sas-phy7-0x4433221107000000-lun-0-part2 \
/       ext3   acl,user_xattr  1 1
proc                 /proc                proc       defaults              0 0
sysfs                /sys                 sysfs      noauto                0 0
debugfs              /sys/kernel/debug    debugfs    noauto                0 0
usbfs                /proc/bus/usb        usbfs      noauto                0 0
devpts               /dev/pts             devpts     mode=0620,gid=5       0 0
```

b. Edit the `/mnt/etc/fstab` file, changing `phy7` to `phy6` device names to reference the backup drive. In the following example, the backup drive is `phy6:1-....`

```
smw:~ # vi /mnt/etc/fstab
smw:~ # cat /mnt/etc/fstab
/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part1 \
swap    swap   defaults        0 0
/dev/disk/by-path/pci-0000:05:00.0-sas-phy6-0x4433221106000000-lun-0-part2 \
/       ext3   acl,user_xattr  1 1
proc                 /proc               proc       defaults             0 0
sysfs                /sys                sysfs      noauto               0 0
debugfs              /sys/kernel/debug   debugfs    noauto               0 0
usbfs                /proc/bus/usb       usbfs      noauto               0 0
devpts               /dev/pts            devpts     mode=0620,gid=5      0 0
```

4. Unmount the backup drive.

   ```
   smw:~ # umount /mnt
   ```

   The SMW can now be shut down and rebooted. Upon display of the **Please select boot device** prompt, select the **BACK-UP DRIVE - SLES 11** entry to boot the backup root partition.

7.9 Desk-side SMW: Set Up the Bootable Backup Drive as an Alternate Boot Device

The following procedure modifies a bootable backup drive for a desk-side SMW in order to boot from and run the desk-side SMW from the backup root partition.

Important: To boot from this backup drive, the primary boot drive must still be operable and able to boot the `grub` boot blocks installed. If the backup drive is modified to boot as an alternate boot device, it will no longer function as a bootable backup if the primary drive fails.

Procedure 61. Desk-side SMW: Set up the bootable backup drive as an alternate boot device

Note: This procedure will **not** provide a usable backup drive that can be booted in the event of a primary drive failure.

Caution: The disk device names shown in this procedure are provided as examples only. Substitute the correct disk devices for your system. For example, on an SMW with three SMW disks, the boot disk is `/dev/sda` and the bootable backup disk is `/dev/sdc`; on an SMW with two SMW disks, the boot disk is `/dev/sda` and the bootable backup disk is `/dev/sdb`.

1. Mount the backup drive's root partition.

   ```
   smw:~ # mount /dev/sdX2  /mnt
   ```

2. Create a new boot entry in the `/boot/grub/menu.lst` file. This entry should be a duplicate of the primary boot entry with the following changes:

   - Modify the title to uniquely identify the backup boot entry.
   - Modify the `root (hd0,1)` directive to reflect the `grub` name of the backup drive. If you do not know the `grub` name of the backup drive, it is provided in the `/boot/grub/device.map` file on the primary drive.
   - Modify the `root=` and `resume=` specifications to reference the backup drive device.

   An example `/boot/grub/menu.lst` file follows. Note the new entry at the end of the file. This example references `/dev/sda` as the primary drive and `/dev/sdc` as the backup drive.

```
smw:~ # cat /boot/grub/menu.lst
# Modified by YaST2. Last modification on Wed Dec 9 15:09:52 UTC 2009
default 0
timeout 8
##YaST - generic_mbr
gfxmenu (hd0,1)/boot/message
##YaST - activate

###Don't change this comment - YaST2 identifier: Original name: linux###
title SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default root=/dev/sda2 \
    resume=/dev/sda1 splash=silent crashkernel=256M-:128M@16M showopts vga=0x31a \
    initrd /boot/initrd-2.6.27.19-5-default

###Don't change this comment - YaST2 identifier: Original name: failsafe###
title Failsafe -- SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default root=/dev/sda2 showopts \
    ide=nodma apm=off noresume edd=off powersaved=off nohz=off highres=off \
    processor.max_cstate=1 x11failsafe vga=0x31a \
    initrd /boot/initrd-2.6.27.19-5-default

###Don't change this comment - YaST2 identifier: Original name: floppy###
title Floppy
    rootnoverify (fd0)
    chainloader +1

### New entry allowing a boot of the back-up drive when the primary drive
### is still present.
title BACK-UP DRIVE - SUSE Linux Enterprise Server 11 - 2.6.27.19-5
    root (hd0,1)
    kernel /boot/vmlinuz-2.6.27.19-5-default root=/dev/sdc2 \
    resume=/dev/sdc1 splash=silent crashkernel=256M-:128M@16M showopts vga=0x31a \
    initrd /boot/initrd-2.6.27.19-5-default
```

3. Modify the backup drive's `/etc/fstab` file to reference the secondary drive slot rather than the first drive slot.

a. Examine the backup drive's `fstab` file.

```
smw:~ # cat /mnt/etc/fstab
/dev/sda1   swap                swap       defaults              0 0
/dev/sda2   /                   ext3       acl,user_xattr        1 1
proc        /proc               proc       defaults              0 0
sysfs       /sys                sysfs      noauto                0 0
debugfs     /sys/kernel/debug   debugfs    noauto                0 0
usbfs       /proc/bus/usb       usbfs      noauto                0 0
devpts      /dev/pts            devpts     mode=0620,gid=5       0 0
```

b. Edit the `/mnt/etc/fstab` file, changing `/dev/sda1` and `/dev/sda2` to reference the backup drive. In the following example, the backup drive is `/dev/sdc`.

```
smw:~ # vi /mnt/etc/fstab
/dev/sdc1   swap                swap       defaults              0 0
/dev/sdc2   /                   ext3       acl,user_xattr        1 1
proc        /proc               proc       defaults              0 0
sysfs       /sys                sysfs      noauto                0 0
debugfs     /sys/kernel/debug   debugfs    noauto                0 0
usbfs       /proc/bus/usb       usbfs      noauto                0 0
devpts      /dev/pts            devpts     mode=0620,gid=5       0 0
```

4. Unmount the backup drive.

   ```
   smw:~ # umount /dev/sdX2
   ```

   The SMW can now be shut down and rebooted. Upon display of the **Please select boot device** prompt, select the **BACK-UP DRIVE - SLES 11** entry to boot the backup root partition.

7.10 Archiving the SDB

The service database (SDB) can be archived by using the `mysqldump` command. For more information, see http://www.dev.mysql.com/doc/.

7.11 Backing Up Limited Shared-root Configuration Data

Cray recommends that if you cannot make a full copy, make a backup copy of the `.shared` root structure before making significant changes to the shared root. You can use the `xtoparchive` utility or the Linux utilities (`cp`, `tar`, `cpio`) to save the shared-root file system. Run these procedures from the boot node.

7.11.1 Using the xtoparchive Utility to Archive the Shared-root File System

Use the xtoparchive command to perform operations on an archive of shared root configuration files. Run the xtoparchive command on the boot node using the xtopview utility in the default view. The archive is a text-based file similar to a tar file and is specified using the required archivefile command-line argument. The xtoparchive command is intended for configuration files only. Binary files will not be archived. If a binary file is contained within a specification file list, it will be skipped and a warning will be issued.

Example 96. Using the xtoparchive utility to archive the shared-root file system

Use the following xtoparchive command to add files specified by the specifications listed in specfile to the archive file archive.042208; create the archive file if it does not already exist:

```
% xtoparchive -a -f specfile archive.042208
```

Note: To archive any specialized files that have changed, invoke the archive_etc.sh script. You can do this while your system is booted or from the boot root and shared root in a system set that is not booted. The archive_etc.sh script uses the xtoprdump and xtoparchive commands to generate an archive of specialized files on the shared root. For more information about archiving and upgrading specialized files, see the shared_root(5), xtoparchive(8), xtopco(8), xtoprdump(8), and xtoprlog(8) man pages.

7.11.2 Using Linux Utilities to Save the Shared-root File System

Use the Linux utilities (cp, tar, cpio) to save the shared-root file system.

Procedure 62. Backing up limited shared-root configuration data

Cray recommends that if you cannot make a full copy, make a backup copy of the .shared root structure before making significant changes to the shared root. Run this procedure from the boot node.

1. Change to the shared root directory that you are backing up.

   ```
   boot:/rr # cd /rr/current
   ```

2. Create a tar file for the directory.

   ```
   boot:/rr/current # tar czf /rr/dot_shared-20120929.tgz .shared
   ```

3. Change to the /rr directory.

   ```
   boot:/rr/current # cd /rr
   ```

4. Verify that the file exists.

```
boot:/rr # ls -al dot_shared-20120929.tgz
-rw-r--r--    1 root     root      7049675 Sep 29 14:21
dot_shared-20050929.tgz
boot:/rr #
```

For more information, see the `cp`(1), `tar`(1), and `cpio`(1) man pages.

7.12 Backing Up Boot Root and Shared Root

Before you back up your boot root and shared root, consider the following issues.

- You must be `root` to do this procedure.
- Do not have file systems mounted on the SMW and the Cray system at the same time.
- File system device names may be different at your site.
- If the backup file systems have not been used yet, you may need to run `mkfs` first.
- File systems should be quiescent and clean (`fsck`) to get an optimal dump and restore.

You can back up the boot root and the shared root by using the `xthotbackup` command or by using the Linux `dump` and `restore` commands.

7.12.1 Using the xthotbackup Command to Back Up Boot Root and Shared Root

Execute the xthotbackup command on the SMW to create a bootable backup. The xthotbackup command must be executed with root privileges. The system set labels in /etc/sysset.conf define disk partitions for backup and source system sets which are used by xthotbackup to generate the appropriate dump and restore commands. The entire contents of the disk partitions defined in a source system set are copied to the corresponding disk partitions in the backup system set. The backup and source system sets must be configured with identical partitions. (Follow the steps provided on the xthotbackup(8) man page in the Initial Setup section to set up identical system sets.) The disk partitions in the backup system set are formatted prior to the dump and restore commands.

Important: The xthotbackup utility can also work with Logical Volume Manager (LVM) volumes, but this requires extra configuration before LVM snapshots can be created. For more information on LVM configuration and xthobackup use with LVM, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) and the xthotbackup(8) man page.

Note: By default, xthotbackup forces file system checks by using fsck -f, unless you use the xthotbackup -n option. All fsck activity is done in parallel (by default, xthotbackup uses the fsck -p option to check all file systems in parallel), unless you use the xthotbackup -l option.

Load the cray-install-tools module to access the xthotbackup utility and the xthotbackup(8) man page.

Example 97. Using the xthotbackup command to create a bootable backup system set

Enter the following to dump all of the partitions from the source label, BLUE, to the backup label, GREEN, and then make them bootable.

Warning: Do not use the xthotbackup command when either the source or destination system set is booted. Running xthotbackup with a booted system set or partition could cause data corruption.

```
smw:~ # xthotbackup -a -b BLUE GREEN
```

The xthotbackup command can also be used to copy selected file systems from source to the backup system set.

Example 98. Using the xthotbackup command to copy selected file systems from source to the backup system set

The following example dumps only the SDB and SYSLOG partitions in the system set labelled BLUE to the system set labelled GREEN.

Warning: Do not use the `xthotbackup` command when either the source or destination system set is booted. Running `xthotbackup` with a booted system set or partition could cause data corruption.

```
smw:~ # xthotbackup -f SDB,SYSLOG BLUE GREEN
```

7.12.2 Using dump and `restore` Commands to Back Up Boot Root and Shared Root

Procedure 63. Backing up the boot root and shared root using the dump and `restore` commands

1. Verify that the Cray system is halted.
2. Open a root session.

   ```
   crayadm@smw:~> su -
   ```

3. Mount the boot root to the SMW.

   ```
   smw:~ # mount /dev/sda1 /bootroot0
   ```

4. Mount the backup boot root to the SMW.

   ```
   smw:~ # mount /dev/sdb1 /bootroot1
   ```

5. Change directories to the backup boot root.

   ```
   smw:~ # cd /bootroot1
   ```

6. Dump and restore boot root to the backup boot root.

   ```
   smw:/bootroot1 # dump -0 -f - /bootroot0 | restore -rf -
   ```

7. When the dump is complete, unmount both boot-root file systems.

   ```
   smw:/bootroot1 # cd /
   smw:/ # umount /bootroot0 /bootroot1
   ```

8. Mount the shared root to the SMW.

   ```
   smw:/ # mount /dev/sdc6 /sharedroot0
   ```

9. Mount the backup shared root to the SMW.

   ```
   smw:/ # mount /dev/sdg6 /sharedroot1
   ```

10. Change directories to the backup shared root.

    ```
    smw:/ # cd /sharedroot1
    ```

11. Dump and restore shared root to the backup shared root.

    ```
    smw:/sharedroot1 # dump -0 -f - /sharedroot0 | restore -rf -
    ```

12. When the dump is complete, unmount both shared root file systems.

```
smw:/sharedroot1 # cd /
smw:/ # umount /sharedroot0 /sharedroot1
```

13. Exit the root session.

```
smw:~ # exit
```

7.13 Backing Up User Data

Backing up user data is a site-specific activity. You can use Linux utilities to back up user files and directories.

7.14 Rebooting a Stopped SMW

If the SMW is down or being rebooted (that is, not fully working), the blade controllers will automatically throttle the high-speed network because they are no longer hearing SMW heartbeats. This is done in order to prevent possible network congestion, which normally requires the SMW to be up in order to respond to such congestion. Once the SMW is up again, the blade controllers will unthrottle the network. (No attempt is made to prevent loss of data or to carry out operations that occur when the SMW is offline.) The consequences of throttling are that the network will perform much more slowly than normal.

When the SMW comes up, it restarts, establishes communications with all external interfaces, restores the proper state in the state manager, and continues normal operation without user intervention.

For a scheduled or unscheduled shutdown and reboot of the SMW, it is necessary to have uncorrupted configuration files on a local SMW disk.

Procedure 64. Rebooting a stopped SMW

1. Verify that your configuration files contain the most recent system configuration.
2. Boot the SMW.

7.15 SMW Recovery

Procedure 65. SMW primary disk failure recovery

The following procedure describes how to recover an SMW primary disk failure. To find out how to create a System Management Workstation (SMW) bootable backup drive, see Procedure 59 on page 239. To find out how to modify a bootable backup drive, in order to boot from and run the SMW from the backup root partition, see Procedure 60 on page 246.

Caution: Booting from the bootable backup disk is intended only for emergency use in the event of failure or loss of data on the primary disk.

To recover an SMW, you must reorder the drives at the front of the SMW. No BIOS or software configuration changes are required.

1. Shutdown the OS on the SMW, if possible.
2. Power the SMW off.
3. Unplug the power cord.
4. For a desk-side SMW, open the disk drive access door, which is on the front of the SMW.
5. Remove the primary disk from its slot. For a desk-side SMW, the primary disk is located at the bottom of the column of disk drives at the front of the SMW. For a Rack-mount SMW, remove the disk drive that is in slot 0.
6. Remove the bootable backup disk and place it in the primary disk slot.
7. Press the reset button (front), if required.
8. Boot the SMW.

7.16 Restoring the HSS Database

When your execute the `xtdiscover` command, it automatically makes a backup copy of partition information, the entire HSS database, and the `/etc/hosts` file. In the event that the HSS database is lost or corrupted for any reason, such as a disk failure, you can restore the HSS database.

Procedure 66. Restoring the HSS database

- Execute these commands on the SMW.

```
smw:~> rsms stop
smw:~> mysql -uhssds -phssds < /home/crayadm/hss_db_backup/database_backup_file
smw:~> cp /home/crayadm/hss_db_backup/hosts_backup_file /etc/hosts
smw:~> rsms start
```

7.17 Recovering from Service Database Failure

If you notice problems with the SDB, for example, if commands like `xtprocadmin` do not work, restart the service-node daemons.

Example 99. Recovering from an SDB failure

Type the following command on the SDB node:

```
sdb:~ # /etc/init.d/sdb restart
```

Commands in this file stop and restart MySQL.

7.17.1 Database Server Failover

The SDB uses dual-ported local RAID to store files.

If you have SDB node failover configured, one service processor acts as the primary SDB server. If the primary server daemon dies, or the node on which it is running dies, the secondary (backup) SDB server that connects to the same RAID storage starts automatically. IP failover directs all new TCP/IP connections to the server, which now becomes the primary SDB server. Connections to the failed server are ended, and an error is reported to the client.

7.17.2 Rebuilding Corrupted SDB Tables

The boot process creates all SDB tables except the accounting and boot tables. If you notice a small corruption and you do not want to reboot, you can change the content of a database table manually by using the tools in Table 7. If you cannot recover a database table in any other way, as a last resort reboot the system.

7.18 Using Persistent SCSI Device Names

Important: The information provided in this section does **not** apply to SMW disks.

SCSI device names (`/dev/sd*`) are not guaranteed to be numbered the same from boot to boot. This inconsistency can cause serious system problems following a reboot. When installing CLE, you **must** use persistent device names for file systems on your Cray system.

Cray recommends that you use the `/dev/disk/by-id/` persistent device names. Use `/dev/disk/by-id/` for the root file system in the `initramfs` image and in the `/etc/sysset.conf` installation configuration file as well as for other file systems, including Lustre (as specified in `/etc/fstab` and `/etc/sysset.conf`). For more information, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

Alternatively, you can define persistent names using a site-specific `udev` rule or `cray-scsidev-emulation`. However, only the `/dev/disk/by-id` method has been verified and tested.

Caution: You must use `/dev/disk/by-id` when specifying the root file system. There is no support in the `initramfs` for `cray-scsidev-emulation` or custom `udev` rules.

7.19 Using a Linux `iptables` Firewall to Limit Services

You can set up a firewall to limit services that are running on your system. Cray has enabled the Linux kernel to provide this capability. Use the `iptables` command to set up, maintain, and inspect tables that contain rules to filter IP packets.

For more information about `iptables`, see the `iptables`(8) man page.

7.20 Handling Single-node Failures

A single-node failure is visible when you use the `xtnodestat` command.

You can parse the `syslog` to look for failures.

If you suspect problems with a node, invoke the `xtcli status` command. Nodes that have failed show an `alert` status. Jobs are not scheduled on the node as long as the alert is set. If problems persist, consult your service representative.

To see cabinet status, use the System Environmental Data Collections (SEDC); see *Using and Configuring System Environment Data Collections (SEDC)* (S–2491).

For more information, see the `xtnodestat`(1), `xtcli`(8), and `xtsedcviewer`(8) man pages.

7.21 Increasing the Boot Manager Time-out Value

On systems of 4,000 nodes or larger, the time that elapses until the boot manager receives all responses to the boot requests can be greater than the default 60-second time-out value. This is due, in large part, to the amount of other event traffic that occurs as each compute node generates its console output.

To avoid this problem, change the `boot_timeout` value in the `/opt/cray/hss/default/etc/bm.ini` file on the SMW to increase the default 60-second time-out value by 60 seconds for every 5,000 nodes; for example:

Example 100. Increasing the `boot_timeout` value

For systems of 5,000 to 10,000 nodes, change the `boot_timeout` line to

```
boot_timeout 120
```

For systems of 10,000 to 15,000 nodes, change the `boot_timeout` line to

```
boot_timeout 180
```

7.22 RAID Failure

System RAID has its own recovery system that the manufacturer supplies. For more information, refer to the manufacturer documentation.

Using the Application Level Placement Scheduler (ALPS) [8]

ALPS (Application Level Placement Scheduler) is the Cray supported mechanism for placing and launching applications on compute nodes. ALPS provides application placement, launch, and management functionality and cooperates closely with third-party batch systems for application scheduling across Cray systems. The third-party batch systems make policy and scheduling decisions, while ALPS provides a mechanism to place and launch the applications contained within batch jobs.

Note: ALPS application placement and launch functionality is only for applications executing on compute nodes. ALPS does not provide placement or launch functionality on service nodes.

8.1 ALPS Functionality

ALPS performs the following functions:

- Assigns application IDs.
- Manages compute node resources by satisfying reservation requests from third-party batch systems.
- Provides a configurable node selection algorithm for placing applications. (See the `ALPS_NIDORDER` configuration parameter in `/etc/sysconfig/alps` Configuration File on page 269 for more information.)
- Launches applications.
- Delivers signals to applications.
- Returns `stdout` and `stderr` from applications
- Provides application placement and reservation information.
- Supports batch and interactive workloads.
- Supports huge pages functionality for CNL applications.
- Provides an XML interface for third-party batch-system communication.
- Provides launch assistance to debuggers, such as TotalView.

- Supports application placement of nonuniform numbers of processing elements (PEs) per node, allowing full use of all compute node resources on mixed-node machines.
- Works with the CLE Node Health software to perform application cleanup following the non-orderly exit of an application (see ALPS and Node Health Monitoring Interaction on page 290) and reservation cleanup following each batch job. For additional information about the CLE Node Health software, see Configuring Node Health Checker (NHC) on page 160.

8.2 ALPS Architecture

The ALPS architecture includes the following clients and daemons, each intended to fulfill a specific set of responsibilities as they relate to application and system resource management. The ALPS components use TCP/IP sockets and User Datagram Protocol (UDP) datagrams to communicate with each other. The `apinit` daemon executes on compute nodes. All other ALPS components execute on service nodes (login, SDB, and boot nodes).

ALPS clients (for detailed descriptions, see ALPS Clients on page 263 and the man page for each ALPS client):

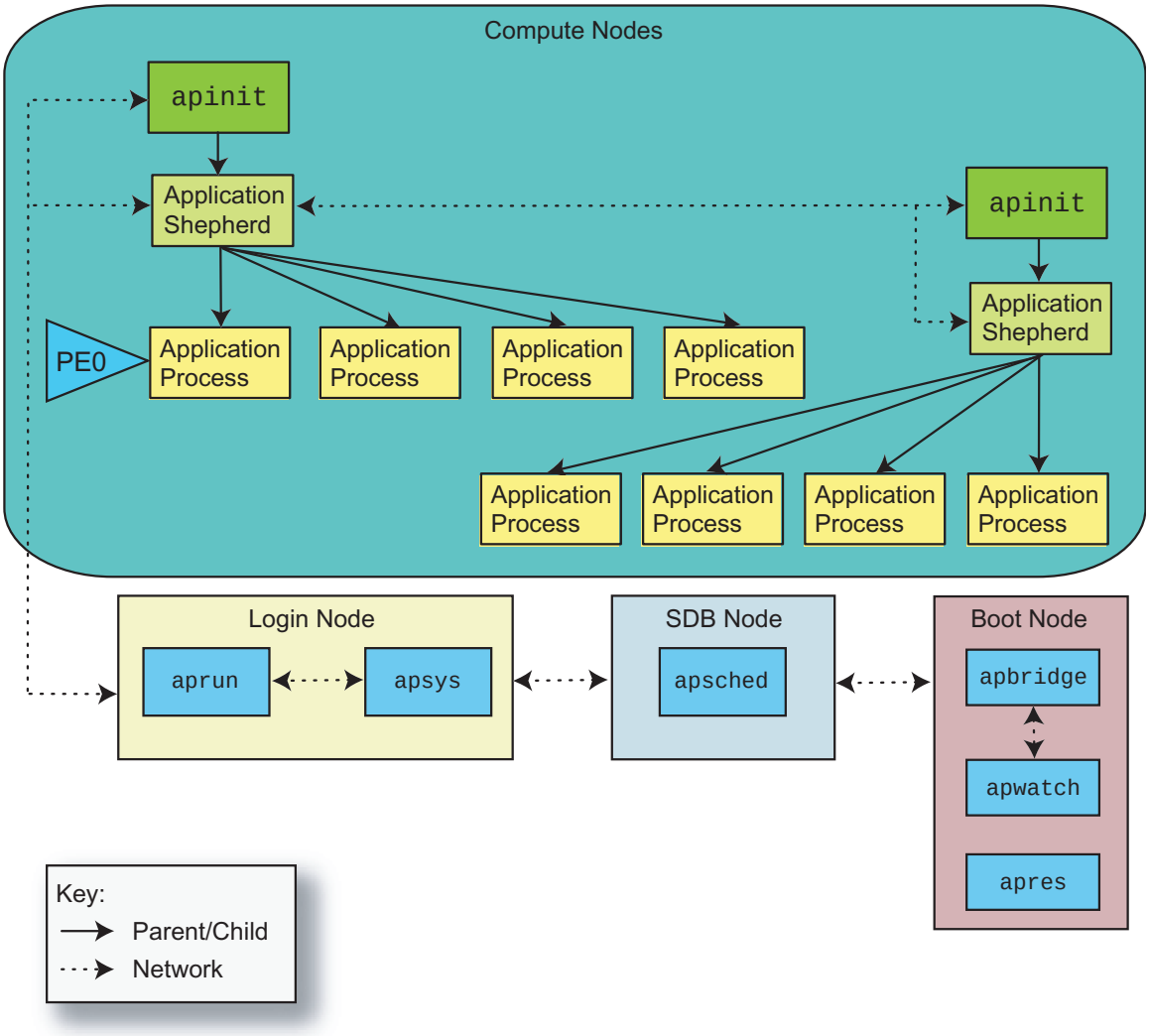
- `aprun`: Application submission
- `apstat`: Application placement and reservation status
- `apkill`: Application signaling
- `apmgr`: Collection of functions usually used by the system administrator in exceptional circumstances to manage ALPS
- `apbasil`: Workload manager interface

ALPS daemons (for detailed descriptions, see ALPS Daemons on page 266 and the man page for each ALPS daemon):

- `apsys`: Client local privileged contact
- `apinit`: Application management on compute nodes
- `apsched`: Reservations and placement decisions
- `apbridge`: System data collection
- `apwatch`: Event monitoring
- `apres`: ALPS database event watcher restart daemon

ALPS uses memory-mapped files to consolidate and distribute data efficiently, reducing the demand on the daemons that maintain these files by allowing clients and other daemons direct access to data they require. Figure 4, illustrates the ALPS processes.

Figure 4. ALPS Process



8.2.1 ALPS Clients

The ALPS clients provide the user interface to ALPS and application management. They are separated into the following distinct areas of functionality: application submission, application and reservation status, application signaling, administrator interface to ALPS, and batch system integration.

8.2.1.1 The aprun Client

The aprun client is used for application submission. Specifically, a user executes the aprun command to run a program across one or more compute nodes. The aprun client serves as the local representative of the application and is the primary interface between the user and an application running on compute nodes. The aprun client parses command-line arguments to determine the application resource requirements. These requirements are submitted locally to apsys, which forwards them to apsched for application placement.

After the application has an assigned placement list of compute nodes, aprun provides application-launch information to the apinit daemon on the first compute node in the placement list. The aprun client also provides user identity and environment information to apinit so that the user's login node session can be replicated for the application on the assigned set of compute nodes. This information includes the aprun current working directory, which must be accessible from the compute nodes.

The aprun client forwards stdin data to apinit, which is delivered to the first processing element (PE) of the application. Application stdout and stderr data is sent from apinit to aprun on the login node.

The aprun client catches certain signals (see the aprun(8) man page) and forwards the signal information to apinit for delivery to the application. Any signal that cannot be caught and that terminates aprun causes apinit to terminate the application.

Note: Do not suspend aprun. It is the local representative of the application that is running on compute nodes. If aprun is suspended, the application cannot communicate with ALPS, such as sending exit notification to aprun that the application has completed.

For more information about using the aprun command, see the aprun(8) man page.

8.2.1.2 The apstat Client

The apstat client reports on application placement and reservation information. It reflects the state of apsched placement decisions. The apstat client does not have dynamic run-time information about an application, so the apstat display does not imply anything about the running state of an application. The apstat display indicates statically that an application was placed and that the aprun claim against the reserved resources has not yet been released.

If no application ID (*apid*) is specified when executing the apstat command, the apstat command displays a brief overview of all applications.

For detailed information about this status information, see the apstat(1) man page.

8.2.1.3 The apkill Client

The apkill client is used for application signaling. It parses the command-line arguments and sends signal information to its local apsys daemon. The apkill command can be invoked on any login or service node and does not need to be on the same node as the aprun client for that application. Based upon the application ID, apsys finds the aprun client for that application and sends the signal to aprun, which sends signal information to apinit for delivery to the application.

The apkill client can send a signal only to a placed application, not a pending application.

For more information about the actions of this client, see the apkill(1) man page and the Linux signal(7) man page.

8.2.1.4 The apmgr Client

The apmgr command is a collection of ALPS-related functions for use by system administrators. These functions (subcommands) often require root permission and are usually used in exceptional circumstances to manage ALPS. The apmgr command is not typically installed on the boot node's file system; it is available on and is run from service nodes.

For information about using the apmgr subcommands, see the apmgr(8) man page.

8.2.1.5 The apbasil Client

The apbasil client is used for batch system integration. It is the interface between ALPS and the batch scheduling system. The apbasil client implements the Batch and Application Scheduler Interface Layer (BASIL). When a job is submitted to the batch system, the batch scheduler uses apbasil to obtain ALPS information about available and assigned compute node resources to determine whether sufficient compute node resources exist to run the batch job.

After the batch scheduler selects a batch job to run, the batch scheduler uses apbasil to submit a resource reservation request to the local apsys daemon. The apsys daemon forwards this reservation request to apsched. If the reservation-request resources are available, specific compute node resources are reserved at that time for the batch scheduler use only.

When the batch job is initiated, the prior confirmed reservation is bound to this particular batch job. Any aprun client invoked from within this batch job can claim compute node resources only from this confirmed reservation.

The batch system uses apbasil to cancel the confirmed reservation after the batch job terminates. The apbasil client again contacts the local apsys daemon to forward the cancel-reservation request to apsched. The compute node resources from that reservation are available for other use after the application has been released.

For additional information, see the apbasil(1) and basil(7) man pages.

8.2.2 ALPS Daemons

ALPS daemons provide support for application submission, placement, execution, and cleanup on the system.

8.2.2.1 The apbridge Daemon

The apbridge daemon collects data about the hardware configuration from the service database (SDB) and sends it to the apsched daemon. It also works with the apwatch daemon to supply ongoing compute node status information to apsched. The apbridge daemon is the bridge from the architecture-independent ALPS software to the architecture-dependent specifics of the underlying system.

The apbridge daemon is not intended for direct use; it is only installed in the boot root and is invoked from within /etc/init.d/alps.

For more information, see the apbridge(8) man page.

8.2.2.2 The apsched Daemon

The apsched daemon manages memory and processor resources of applications running on compute nodes.

Note: Only one instance of the ALPS scheduler can run across the entire system at a time.

When apsched receives a request for application placement from aprun, it either returns a message regarding placement or a message indicating why placement is not possible (errors in the request or temporarily unavailable resources). When an application terminates, an exit message is sent to apsched, and it releases the resources reserved for the application.

The apsched daemon writes a log file on the node on which apsched is executing. By default, this is the SDB node.

For more information, see the apsched(8) man page.

8.2.2.3 The apsys Daemon

The apsys daemon provides a central privileged point of contact and coordination between ALPS components running on login and other service nodes. The apsys daemon receives incoming requests and forks child agent processes to delegate responsibilities and improve scalability and responsiveness. An apsys daemon executes on each login node and writes a log file on each login node.

Each `aprun` client has an `apsys` agent associated with it. Those two programs are on the same login node and communicate with each other over a persistent TCP/IP socket connection that lasts for the lifetime of the `aprun` client. The `apsys` daemon passes `aprun` messages to `apsched` over a transitory TCP/IP socket connection and returns the response to `aprun`.

An `apsys` agent is created to service `apbasil` and `apkill` messages. These programs communicate over transitory TCP/IP socket connections. The `apsys` agent handles the `apkill` message itself and forwards `apbasil` messages to `apsched`.

Each `apsys` agent maintains a separate agents file that is located in the ALPS shared directory. The file name format is `agents.`*nid*, for example, `/ufs/alps_shared/agents.40`. For information about defining the ALPS shared directory, see Configuring ALPS on page 269.

`apsys` also provides a facility to system administrators to execute prologue and epilogue actions for each `aprun` execution. Administrators can specify a script or binary pathname in the `alps.conf` configuration file under the `apsys` block. The variables used to specify these pathnames are `prologPath` and `epilogPath`. `prologTimeout` and `epilogTimeout` are used to enforce timeouts on prologue and epilogue scripts and executables. To see an example of this in `alps.conf`, see Example 102.

For more information, see the `apsys`(8) man page.

8.2.2.4 The `apwatch` Daemon

The `apwatch` daemon waits for events and sends compute node status changes to `apbridge`, which sends it to `apsched`. The `apwatch` daemon is not intended for direct use; it is only installed in the boot root and is invoked from within `/etc/init.d/alps`.

For more information, see the `apwatch`(8) man page.

8.2.2.5 The `apinit` Daemon

The `apinit` daemon launches and manages new applications. A master `apinit` daemon resides on every compute node, initiates all new activity on that node, and writes a log file on the compute node. The `aprun` client connects to the `apinit` daemon on the first node of an application's allocated node set and sends a launch message containing all of the information the compute nodes need to launch and manage the new application.

The `apinit` daemon then forks a child process (referred to as the *apshepherd* or just *shepherd*) and transfers responsibility for managing the application on that node to that child. If the application requires more compute nodes, the shepherd process communicates to the `apinit` daemon on the next compute node, which forks another shepherd child process.

If the application is placed on more than one compute node, ALPS uses a TCP fan-out control tree network for application management messages to do binary transfer of the application when requested, and to handle application `stdin`, `stdout`, and `stderr` data. The root of the fan-out control tree is `aprun`. The width of the fan out is configured within the `/etc/opt/cray/alps/alps.conf` file and is `32`, by default.

The `apinit` daemon is under the control of RCA. If the `apinit` daemon fails, RCA restarts `apinit`. If RCA is unable to restart `apinit` after several attempts, ALPS is notified and the node is made unavailable (`DOWN`) for applications.

For more information, see the `apinit`(8) man page.

8.2.2.6 The apres Daemon

The ALPS `apres` event watcher restart daemon registers with the event router daemon to receive `ec_service_started` events. When the service type is the SDB (`RCA_SVCTYPE_SDBD`), ALPS updates its data to reflect the current values in the SDB. The `apres` daemon is invoked as part of the ALPS startup process on the boot node.

The `apres` daemon is not intended for direct use; it is only installed in the boot root and is invoked from within `/etc/init.d/alps`.

For more information, see the `apres`(8) man page.

8.2.2.7 ALPS Log Files

Each of the ALPS daemons writes information to its log file in `/usr/opt/cray/alps/log` on whichever node that runs the daemon. The name of the log file consists of the daemon name appended with the month and day, such as `apsched0302`.

The `apinit` log file is in the `/usr/opt/cray/alps/log` directory on each compute node and also has a node ID appended to it, such as `apinit0302.00206`. Because this directory is in memory, the `apinit` log file is lost when a compute node is rebooted.

Each system has one `apbridge` daemon, one `apwatch` daemon, and one `apres` daemon, all of which must execute on the same node. By default, this is the boot node. These three daemons write to one log file on that node. The log file name format is `apbridge`*mmdd*, for example, `apbridge1027`.

8.2.2.8 Changing Debug Message Level of apsched and apsys Daemons

The level of debug messages written by the `apsched` and `apsys` daemons is defined in the `/etc/opt/cray/alps/alps.conf` configuration file. You can change the debug level dynamically by modifying the `alps.conf` file and sending a `SIGHUP` signal to `apsched` or `apsys`, as applicable, to read the `alps.conf` file.

8.3 Configuring ALPS

ALPS uses the following three files:

- `/etc/sysconfig/alps` configuration file
- `/etc/opt/cray/alps/alps.conf` configuration file
- `/etc/init.d/alps` file, which is used to start and stop ALPS components and does not require customization

Note: When configuring the RAID LUNs (logical units), verify that write caching is enabled on the LUN that contains the ALPS shared file system. For more information about RAID configuration, see the *Installing Cray System Management Workstation (SMW) Software* (S–2480) and the *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

8.3.1 `/etc/sysconfig/alps` Configuration File

The `/etc/sysconfig/alps` file is in both the boot root and in the shared root. If you defined the ALPS-related parameters in your `CLEinstall.conf` file, after installation the parameters and settings are placed into your `/etc/sysconfig/alps` file.

If you do not define the ALPS-related parameters in your `CLEinstall.conf` file, you must define the parameters in your `/etc/sysconfig/alps` file (required parameters are indicated) and then start the ALPS daemons in order to use ALPS.

Note: When changing parameter settings, update the `/etc/sysconfig/alps` file in **both** the boot root and in the shared root and restart the ALPS daemons on all service nodes.

`ALPS_MASTER_NODE`

(Required) Specifies the node name (`uname -n`) of the service node that runs `apsched`. Cray recommends that the SDB node be used as the `ALPS_MASTER_NODE`. For example:
`ALPS_MASTER_NODE="sdb"`

`ALPS_BRIDGE_NODE`

(Required) Specifies the node name (`uname -n`) of the service node that runs `apbridge`. This is usually the boot node. Network connectivity between the SMW and the `ALPS_BRIDGE_NODE` parameter is required. (Such connectivity is guaranteed to exist from the boot node.) This default value is enforced in the `/etc/init.d/alps` file. For example:
`ALPS_BRIDGE_NODE="boot"`

ALPS_MOUNT_SHARED_FS

Specifies if a separate file system is to be mounted at ALPS startup to hold control data; default is no. For configurations using multiple login nodes, a shared file system is required, and the shared file system must be mounted before ALPS is started. For example: ALPS_MOUNT_SHARED_FS="no"

ALPS_SHARED_DEV_NAME

Specifies the device to mount at ALPS start-up. If it is null and ALPS_MOUNT_SHARED_FS is yes, the device is determined by /etc/fstab. This parameter is not used unless yes is specified for ALPS_MOUNT_SHARED_FS. For example: ALPS_SHARED_DEV_NAME="ufs:/ufs/alps_shared"

ALPS_SHARED_MOUNT_OPTIONS

Specifies the shared mount options. Set this parameter only if ALPS_MOUNT_SHARED_FS is yes and ALPS_SHARED_DEV_NAME is not null. For example: ALPS_SHARED_MOUNT_OPTIONS="-t nfs -o tcp,rw"

ALPS_IP_PREFIX

(Deferred implementation) Use of this parameter has no effect. Specifies the first two octets for IP addresses on the high-speed network (HSN). These are internal addresses within the HSN. For example: ALPS_IP_PREFIX="10.128"

ALPS_NIDORDER

The nid ordering option for apbridge can be defined. Because this is a system-wide setting, Cray recommends that you change this option only when you reboot your system to ensure apbridge and apsched are restarted in the correct sequence. The valid values are:

-On — Leave ALPS_NIDORDER unset, or set this to -On, to assign order of nodes based solely on ascending numerical Node ID (NID) order. This is the default sorting order and the preferred option for Aries systems

-Ox — Assigns node order by maximum dimension as the outer dimension; the smallest dimension will change most quickly. For example, a system with an "xyz"

topology of 6x12x8 would have the *x* dimension changing most rapidly and the *y* dimension changing least rapidly, therefore this option would order them as (0,0,0), (1,0,0), (2,0,0), (3,0,0), (4,0,0), (5,0,0), (0,0,1), (1,0,1), (2,0,1) and so on.

This option often improves application performance over using the `-On` option. Applications that use a small percentage of the machine, especially on machines that are largely cubic in their dimensions, may not benefit from this configuration.

For max-major dimension ordering (minimum dimension varies fastest)

`-Oy` Specifies *y*-major dimension ordering, so that the *y* dimension always varies last. This is best used on Cray XE systems of more than three cabinets.

`-Or` This is the reverse of the `-Ox` ordering method; the order of nodes is assigned by the minimum dimension as the outer dimension. This is usually not recommended.

`-Of` For field ordering. (Uses `od_allocator_id` column.)

`-O2` Assigns order of nodes based on the "2x2x2" NID reordering method, which is a merger of the incidental small node packing of the simple NID number method and the inter-application interaction reduction of the "xyz" method. Cray recommends this option for Cray XE and Cray XK systems, as it produces improved bisection bandwidth of placements as compared to the `-Ox`, `-Oy`, `-Or`, or `-Of` ordering.

`APWATCH_LIBRARY_PATH`

The `LD_LIBRARY_PATH` "add-on" needed for `apwatch`; it includes the path to the `gnet` and `glib` libraries and the `rsms` and `erd` libraries.

For example:

```
APWATCH_LIBRARY_PATH="/opt/gnet/lib:/opt/glib/lib:/opt/cray/librsmsevent.so: \
/opt/cray/libcray_event_router.so:/opt/gnome/lib64"
```

APWATCH_ERD

(Required) The host that has the event router daemon (ERD) running; typically, this is the host name of the SMW.

For example: APWATCH_ERD="smw"

ALPS_X2CONFIG

The directory where some Cray X2 configuration scripts are located. This is used only on Cray X2 systems and should be left unset on all other systems.

APEVENT_SYNC_SECS

The interval in seconds between apevent sync requests.

For example: APEVENT_SYNC_SECS="300"

A separate file system for control data is mounted at ALPS startup. This is assumed to be a mount point. Specify the device to mount at ALPS start-up using the parameter ALPS_SHARED_DEV_NAME. If it is null and ALPS_MOUNT_SHARED_FS is yes, the device is determined by /etc/fstab.

The following example shows a sample /etc/sysconfig/alps configuration file.

Example 101. Sample /etc/sysconfig/alps configuration file

```
## Path:        System/ALPS
## Description: ALPS options
## Type:        string
## Default:     alps
## ServiceReload: alps
## ServiceRestart: alps
#

## Type:        string
## Default:     ""
#
# Nodename (uname -n) of service node that will act as the ALPS
# master -- the node that will run apsched.  In most cases this
# will be the sdb node.  This parameter is required for all
# configurations.
#
ALPS_MASTER_NODE="sdb"

## Type:        string
## Default:     ALPS_MASTER_NODE or ""
#
# Nodename (uname -n) of service node that will act as the ALPS
# "bridge" -- the node that will run apbridge.  In most cases this
# will be the boot node.  This parameter is required for all
# configurations.  If no value is set, then the ALPS_MASTER_NODE
# will be used, but that will work only if there is network
# connectivity between the master node and the SMW.  (Such
# connectivity is guaranteed to exist from the boot node.)
# This default value is enforced when used - in the alps.init script,
# as in: ${ALPS_BRIDGE_NODE:=ALPS_MASTER_NODE}
```

```
# If this variable is read otherwise, the empty string will be found as
# the default value.
#
ALPS_BRIDGE_NODE="boot"

#
## Type:        yesno
## Default:     no
#
# Configurations that use multiple login nodes must use a shared
# filesystem. If "yes", the filesystem indicated by the apsched:sharedDir
# variable in alps.conf will be mounted at ALPS start-up by the ALPS start-up
# scripts.  Note that /ufs mounts are brought up before ALPS is launched, so
# if the alps.conf value is resident on /ufs, this option should be "no".
# Otherwise,  this script will have to mount the alps.conf value and this option
# should be "yes".
#
ALPS_MOUNT_SHARED_FS="no"

#
# MOVED: ALPS_SHARED_DIR_PATH
#
# The value for the ALPS shared directory is now set within alps.conf
#
# alps.conf:
# apsched
#  ...
#  sharedDir /ufs/alps_shared
#  ...
# /apsched
#

## Type:        string
## Default:     ""
## Example:     "ufs:/ufs/alps_shared"
#
# Device to mount at ALPS start-up.  If it is null but ALPS_MOUNT_SHARED_FS
# is "yes", then the device will be determined by /etc/fstab.  This
# parameter is not used unless ALPS_MOUNT_SHARED_FS is "yes".
#
ALPS_SHARED_DEV_NAME=""

## Type:        string
## Default:     ""
## Example:     "-t nfs -o tcp,rw"
#
# This parameter is not used unless ALPS_MOUNT_SHARED_FS is "yes" and
# ALPS_SHARED_DEV_NAME is not null.
#
ALPS_SHARED_MOUNT_OPTIONS=""

## Type:      string
## Default:     "10.168"
## Example:     "10.168"
#
# The first two octets for IP addresses on the high-speed network.
# These are internal addresses within the HSN.
#
ALPS_IP_PREFIX="10.128"

## Type:        string
## Default:
## Example:     -On
#
```

```
# The nid ordering option for apbridge can be defined.
# The choices are:
#   -On or (leave unset) for numerical ordering, i.e. no special order
#       This option (-On) is preferred for Aries systems
#   -Ox for max-major dimension ordering (minimum dimension varies fastest)
#   -Oy for y-major dimension ordering (y dimension varies last)
#   -Or for reverse of max (i.e. min) ordering (usually not recommended)
#   -Of for field ordering (uses od_allocator_id column)
#   -O2 for 2x2x2 ordering for older systems and certain irregular-sized systems;
#         for most Gemini systems it uses an ordering that improves further on
#         the 2x2x2 ordering; this (-O2) is recommended for Gemini sites as
#         it improves bisection bandwidth of placements compared to -Ox/y/r/n
#
ALPS_NIDORDER=""

## Type:      string
## Example:
## "/opt/gnet/lib:/opt/glib/lib:/opt/cray/librsmsevent.so:/opt/cray/libcray_event_router.so"
#
# The LD_LIBRARY_PATH "add-on" needed for APWATCH, it includes the path
# to the gnet and glib libraries and the rsms and erd libraries.
#
APWATCH_LIBRARY_PATH="/opt/gnet/2.0.5/64/lib:/opt/glib/2.4.2/64/lib:/opt/cray/lib64:/opt/gnome/lib64"

## Type:      string
## Default:     ""
## Example:     "whistler-smw"
#
# The host which has the Event Router Daemon running;
# typically this is the hostname of the smw.
# This parameter is required for all configurations.
#
APWATCH_ERD="smw"

## Type:        string
## Default:     "/opt/cray/etc"
## Example:     "/opt/cray/etc"
#
# The directory where some X2 config scripts are located;
# needs to be not empty only on systems w/an X2 component.
#
ALPS_X2CONFIG="/opt/cray/etc"

## Type:      integer
## Default:     "300"
## Example:     "60"
#
# The interval in seconds between apevent sync requests
#
APEVENT_SYNC_SECS="300"
```

8.3.2 The `alps.conf` Configuration File

The `/etc/opt/cray/alps/alps.conf` file is in the shared root and boot root and contains ALPS static configuration information used by the `apsched` and `apsys` daemons and the `aprun` and `apstat` commands. This file also contains application cleanup and reservation cleanup parameters.

Note: You can change the parameter settings dynamically by modifying the `alps.conf` file and sending a `SIGHUP` signal to `apsched` or `apsys`, as applicable, to reread the `alps.conf` file.

Example 102. Sample `/etc/opt/cray/alps/alps.conf` configuration file

```
#
# (c) 2013 Cray Inc.  All Rights Reserved.  Unpublished Proprietary
# Information.  This unpublished work is protected to trade secret,
# copyright and other laws.  Except as permitted by contract or
# express written permission of Cray Inc., no part of this work or
# its content may be used, reproduced or disclosed in any form.
#

# ALPS configuration file
# See the system admin guide for more information
# on possible settings and values

# Config values in this section affect all ALPS daemons
logging
# log method: 1 - traditional log files 2 - llm logging 3 - both
 logMethod 1
/logging

apsched
 alloc  0
 bridge  1
 fanout  32

# Debug level - a bitmask of what debug info to log:
# 0x0001 - General debug
# 0x0002 - Placement function details
# 0x0004 - sendMsg debug
# 0x0008 - Check listen port details
# 0x0010 - Recovery debug
# 0x0020 - XT optimized placement
# 0x0040 - Configuration debug
# 0x0080 - Context switch debug
# 0x0100 - Protection domain debug
# 0x0200 - Reserved for future use
# 0x0400 - GNI (Gemini/Aries) code debug
# 0x0800 - State refresh debug
# 0x1000 - Timings for reservations/appinfo file I/O
 debug  0
# Default CPU affinity: values cpu (default), none, numa, depth
# cpuAffinity cpu
# Default lustre cache flushing at app exit: values 0, 1
#       lustreFlush 1
# Default memory compaction at app exit: values 0, 1
#       memoryCompact   1
```

```
# Default app node share mode for cores and memory: values exclusive, share
# nodeShare exclusive
# Max number of simultaneous reservations
# maxResv  4096
# Default number of CPUs Per Compute Unit for batch/interactive
# (default is 0 on Gemini, 1 on Aries)
# batchCPCU 0
# interactiveCPCU 0
# If set, reservations will get all node resources
# (used for systems with hyperthreading)
# resFullNode 1
# Maximum concurrent claims (apruns) allowed per reservation
#       claimsPerRes    1000

# Variables to control suspend/resume
# Enable suspend/resume code
#       suspendResume   1
# Max # of apps/node (default 4; must be 2 or 4)
#       loadLimit       4
# Number (e.g. 32) or size (e.g. 2G) of compute node fake numa nodes
# This must be identically configured on the compute nodes
# via kernel parameter 'numa=fake=<num>' or 'numa=fake=<siz>'
#       fakeNumaNodes   32
#       fakeNumaNodes   1G

# Values used for systems with Gemini or Aries network:
# - nttSize: if set to 0, the apps will NOT use the NTT
#   (and on Gemini, all apps will get a global PTag)
# nttSize  0

# Values used for systems with Gemini network:
# - pTagGlobalNodes: if set to X, apps >= than X nodes
#   will use a global PTag (and NOT use the NTT)
# pTagGlobalNodes 5000
# - pTagFreeDelay: delay freeing PTags for X seconds
 pTagFreeDelay 30

# Value used for systems with Aries network:
# - pKeyFreeDelay: delay freeing PKeys for X seconds
 pKeyFreeDelay 30

#
# Variables to enable user/system protection domains
#
# Max # of user protection domains allowed
# pDomainMax 10
# List of system protection domains used by all applications
# pDomainIDs sys1,sys2

#
# Directory containing shared files
#
 sharedDir      /ufs/alps_shared
/apsched

apsys

# Debug level - a bitmask of what debug info to log:
```

```
# 0x0001 - General debug
# 0x0002 - More detailed debug
# 0x0004 - Periodic messages issued in poll loops
# 0x0008 - Accounting debug messages
# 0x0010 - Reserved for future use
# 0x0020 - Timing of msync()/write() calls
 debug  1
# aruEnable - Y or Yes generates Application Resource Utilization (ARU) files
# aruEnable Yes
# aruPath - file name template for ARU file
# specified name will have the apid appended when the file is created
# ignored unless aruEnable is set to Y or Yes
# aruPath  /tmp/arufile
# prologPath - location of the executable file to be run before application
#       prologPath /usr/local/adm/sbin/prolog
# epilogPath - location of the executable file to be run after application
#       epilogPath /usr/local/adm/sbin/epilog
# prologTimeout - time in seconds before prolog is aborted as "hung"
#       prologTimeout 10
# epilogTimeout - time in seconds before epilog is aborted as "hung"
#       epilogTimeout 10
/apsys

aprun
# each of the aprun settings creates an association between an ALPS value
# and a WLM environment variable that will be defined on the compute nodes
# defineNid SLURM_NODEID
# defineEachID SLURM_PROCID
# defineNPPN SLURM_TASKS_PER_NODE
# defineLocalEnt SLURM_LOCALID
/aprun

apstat
# nodeTable NID,Arch,State,HW,Rv,Pl,PgSz,Avl,Conf,Placed,PEs,Apids
 nodeTable NID,Arch,State,CU,Rv,Pl,PgSz,Avl,Conf,Placed,PEs,Apids
# appsTable Apid,ResId,User,PEs,Nodes,Age,State,Command
# resvTable ResId,ApId,From,Arch,PEs,N,d,Memory,State
# pendingAppsTable Pid,User,w:d:N,Nid,Age,Command,Why
# gpuTable NID,Module,State,Memory(MB),Family,ResId
# Gemini pDomainTable
# pDomainTable PDomainID,Type,Uid,PTag,Cookie
# Aries pDomainTable
# pDomainTable PDomainID,Type,Uid,Cookie,Cookie2
/apstat

#
# Configure application and reservation cleanup
#
# configured - on or off (default = on)
# reports - save information about ongoing cleanups for apstat -C (on or off)
# reportWait - time (ms) to wait for node health before writing a report
# iterationSleep - time (ms) to pause between cleanup iterations
# iterationMax - maximum cleanup iterations to execute before giving up
# connectTimeout - time (ms) to allow an inter-node connection to be established
# connectAttempts - maximum inter-node connection attempts
# waitMin - minimum time (ms) to give any cleanup iteration to complete.  The
#           maximum wait time is calculated from this and other cleanup
#           parameters.
```

```
#
cleanup
 application
  reports  on
  reportWait 2000
  iterationSleep 1000
  iterationMax 10
  connectTimeout 1000
  connectAttempts 5
  waitMin  5000
 /application
 reservation
  configured on
  reports  on
  reportWait 2000
  iterationSleep 1000
  iterationMax 10
  connectTimeout 1000
  connectAttempts 5
  waitMin  5000
 /reservation
/cleanup
```

Procedure 67. Releasing a reserved system service protection domain

1. Ensure that no applications are running or reboot the system.
2. Remove the desired identifier from the `/etc/opt/cray/alps/alps.conf` `pDomainIDs` list.
3. Send a `SIGHUP` signal to `apsched` so that it restarts and reads the new `pDomainIDs` list.

8.3.2.1 Global Configuration Parameters

These configuration parameters affect all ALPS daemons.

`logMethod` Select `1` to generate traditional log files, `2` to do `llm` logging, or `3` to do both.

8.3.2.2 apsched Configuration Parameters

These configuration parameters affect `apsched` only.

`alloc` If this field is set to `0` or is not specified, the distinction between batch and interactive nodes is enforced. If this field is set as nonzero, no distinction is made by ALPS; job schedulers will likely still limit their placement only to nodes marked as batch.

Default: 0

bridge
: Enables the apbridge daemon to provide dynamic rather than static information about the system node configuration to apsched. Cray strongly recommends setting the bridge parameter to use the apbridge daemon.

  Default: 1 (enabled)

fanout
: This value controls the width of the ALPS TCP/IP network fan-out tree used by apinit on the compute nodes for ALPS application launch, transfer, and control messages.

  Default: 32

debug
: This field is set to a default level of 0 for apsched and 1 for apsys. For information about valid values, see the apsched(8) and apsys(8) man pages.

  Default: 0 (apsched), 1 (apsys)

cpuAffinity
: Supports switchable default CPU affinity in apsched. Valid values are cpu, depth, numa, and none. If the user has not explicitly set the CPU affinity using the aprun -cc option, the aprun command checks for and uses the CPU affinity from apsched; if there is not a value from apsched, it uses the default value. For more information, see the aprun(1) man page.

  Default: cpu

lustreFlush
: Supports switchable default Lustre cache flushing as part of application exit processing on the compute nodes. Enabling this Lustre cache flushing provides more consistent application run times. When Lustre cache flushing is enabled, all of the Lustre cache flushing completes as part of the application exit processing. The next application executing on the same set or subset of compute nodes no longer inherits a variable amount of run time due to Lustre cache flushing from a previous application.

  Valid values are 0 (disabled) and 1 (enabled). Apsched provides this lustreFlush value to the apinit daemon to enable or disable Lustre cache flushing as part of application exit processing.

  Note: This value cannot be set on an individual application basis. It is a system-wide setting.

Default: 1 (enabled)

memoryCompact

Supports switchable memory compaction at application exit. Valid values are 0 (disabled) and 1 (enabled).

Default: 1 (enabled)

nodeShare Controls which compute node cores and memory are put into an application cpuset on the compute node. The valid values are exclusive and share.

The exclusive setting puts all of a compute node's cores and memory resources into an application-specific cpuset on the compute node. This allows the application access to any and all of the compute node cores and memory. This can be useful when specifying a particular CPU affinity binding string through the aprun -cc option.

The share setting restricts the application specific cpuset contents to only the application reserved cores and memory on NUMA node boundaries. That is, if an application requests and is assigned cores and memory on NUMA node 0, only NUMA node 0 cores and memory will be contained within the application cpuset. The application will not have access to the cores and memory on other NUMA nodes on that compute node.

To override the default system-wide setting in /etc/opt/cray/alps/alps.conf on an individual basis, use the aprun -F option. For more information, see the aprun(1) man page.

Default: exclusive

maxResv The maximum number of simultaneous reservations allowed.

Default: 4096

batchCPCU Specifies the default number of CPUs per compute unit for nodes whose allocation mode attribute is marked as batch.

Defaults: 0 (Gemini systems); 1 (Aries systems)

interactiveCPCU

Specifies the default number of CPUs per compute unit for nodes whose allocation mode attribute is marked as interactive.

Defaults: `0` (Gemini systems); `1` (Aries systems)

`resFullNode`

When set to a non-zero value, reservations will get all node resources (threads) regardless of other reservation parameters. The other reservation parameters will be taken into account to determine how many nodes are needed to accommodate the reservation.

For example, a reservation with a `width` of 8 and an `nppn` of 4 will reserve two nodes. When `resFullNode` is not set, ALPS would limit the user to launching only four instances of the application on each node. With `resFullNode` set, ALPS will still reserve two nodes but it will reserve all the hyper-threads and associated resources on both nodes. Also, the user would be allowed to launch as many application instances as there are hyper-threads.

Default: `1` (enabled)

`claimsPerRes`

Maximum number of concurrent claims (`aprun` instances) allowed per reservation.

Default: `1000`

`suspendResume`

Determines whether suspend/resume is enabled. Valid values are `0` (disabled), `1` (enabled in Phase 2 mode), and `-1` (enabled in Phase 1 mode). Suspend/resume enables applications to oversubscribe compute node CPUs; BASIL provides a switching mechanism that enables one job to suspend another job on the node. In Phase 1 mode a maximum of two co-resident jobs is permitted; in Phase 2 mode, this limit is raised to four.

`loadLimit` Determines the maximum number of applications per node permitted if suspend/resume is enabled. Valid values are `2` or `4`.

Default: 4

fakeNumaNodes

Determines the number or size of compute node fake NUMA nodes. If set as an integer, this is interpreted as the number; if set as a string (i.e., 2G), this is interpreted as a size. This must be configured identically on the compute nodes via the kernel parameter numa=fake=*number* or numa=fake=*size*.

nttSize — If set to 0, applications will not use the NTT (Network Translation Table). This parameter is used primarily for testing purposes.

pTagGlobalNodes

The minimum application size in nodes that forces ALPS to assign a global pTag value. If set, applications placed on a number of nodes equal to or greater than this value will use a global pTag, and not the NTT. This value is used on Gemini systems only.

pTagFreeDelay

Specifies the number of seconds to delay allocation of a pTag used previously. (The current PTag allocation implementation rotates through the pTags range.) This value is used on Gemini systems only.

Default: 30 seconds

pKeyFreeDelay

Specifies the number of seconds to delay freeing a pKey used previously. This value is used on Aries systems only.

Default: 30 seconds

pDomainMax — Specifies the maximum number of concurrent preallocated protection domains allowed. The absolute maximum is 256 and apsched will silently truncate higher values to 256.

Default: 0 (disabled)

pDomainIDs — A comma-separated list of one or more system service dedicated protection domain identifiers.

sharedDir — (Required) Specifies the directory path to the file that contains ALPS control data. If ALPS_MOUNT_SHARED_FS is set to yes, this is assumed to be a mount point.

Default: /ufs/alps_shared

8.3.2.3 apsys Configuration Parameters

These configuration parameters affect `apsys` only.

`debug` This field is set to a default level of `1` for both `apsched` and `apsys`. For information about valid values, see the `apsched`(8) and `apsys`(8) man pages.

Default: `1`

`aruEnable` Set to `Y` to generate Application Resource Utilization (ARU) files.

Default: not set

`aruPath` Specify the path name template for the ARU file. The `apid` will be appended to the specified file name when the file is created. This value is ignored if `aruEnable` is not set to `Y`.

`prologPath` Specifies the location of a file to be executed before an application is launched. If this variable is not specified, nothing will be run before the `aprun` commands.

Default: not set

`epilogPath` Specifies the location of a file to be executed after an application exits. If this variable is not specified, nothing will be run after the `aprun` commands.

Default: not set

`prologTimeout`

Specifies the maximum number of seconds to allow for the prolog to run. If it runs longer, the prolog will be terminated and an error will be returned. If this variable is not specified, the prolog will be given unlimited time to complete.

Default: not set

`epilogTimeout`

Specifies the maximum number of seconds to allow for the epilog to run. If it runs longer, the epilog will be terminated. If this variable is not specified, the epilog will be given unlimited time to complete.

Default: not set

8.3.2.4 aprun Configuration Parameters

These configuration parameters affect aprun only. Each of these settings creates an association between an ALPS value and a SLURM workload manager environment variable that will be defined on the compute nodes.

defineNid Corresponding environment variable: SLURM_NODEID

defineEachID

Corresponding environment variable: SLURM_PROCID

defineNPPN Corresponding environment variable: SLURM_TASKS_PER_NODE

defineLocalEnt

Corresponding environment variable: SLURM_LOCALID

8.3.2.5 apstat Configuration Parameters

These values can be used to customize the apstat reports. Each configuration parameter is followed by a comma-separated list defining the data elements to be included in the report. The data elements are described in the apstat(1) man page.

nodeTable Default configuration:
NID,Arch,State,HW,Rv,Pl,PgSz,Avl,Conf,Placed,PEs,Apids

appsTable Default configuration:
Apid,ResId,User,PEs,Nodes,Age,State,Command

resvTable Default configuration:
ResId,ApId,From,Arch,PEs,N,d,Memory,State

pendingAppsTable

Default configuration:
Pid,User,w:d:N,Nid,Age,Command,Why

gpuTable Default configuration:
NID,Module,State,Memory(MB),Family,ResId

pDomainTable

Default configuration (Gemini):
PDomainID,Type,Uid,PTag,Cookie

Default configuration (Aries):
PDomainID,Type,Uid,Cookie,Cookie2

8.3.2.6 Application and Reservation Cleanup Configuration Parameters

The application and reservation cleanup functions have similar configuration value fields but can be configured separately.

`configured` (Reservation cleanup only)

On or off. (Application cleanup is always on.)

Default: `on`

`reports` If on, saves information about ongoing cleanups for use by the `apstat -C` command.

Default: `on`

`reportWait` The time in milliseconds to wait for node health before writing a report.

Default: `2000`

`iterationSleep`

The time in milliseconds to pause between cleanup iterations.

Default: `1000`

`iterationMax`

The maximum number of cleanup iterations to execute before giving up.

Default: `10`

`connectTimeout`

The time in milliseconds to allow an inter-node connection to be established.

Default: `1000`

`connectAttempts`

The maximum number of inter-node connection attempts to execute before giving up.

Default: `5`

`waitMin` The minimum time in milliseconds to give any cleanup iteration to complete. The maximum wait time is calculated from this and the other cleanup parameters.

Default: `5000`

8.3.2.7 Using Suspend/Resume

Suspend/resume enables users to oversubscribe compute node CPUs, and provides a mechanism (via the batch system and BASIL) whereby applications can put other applications in a suspended state and take over use of the node. Note that suspend/resume enables users to oversubscribe compute node CPUs, not node memory. The amount of memory available to any given application is inversely proportional to the number of applications placed on the node: in other words, if two applications are placed on a node, each can use no more than half the node's memory.

When suspend/resume is enabled, the default `apsched` behavior changes from "exclusive" to "shared," unless a node reservation is explicitly exclusive; i.e., if the batch system reserves the node in interactive mode. When transitioning to or from suspend/resume, `apinit` clears any hugepage minimum values, and while suspend/resume is enabled, the `aprun -F exclusive`, `-r` *numcores*, `-ss`, and other NUMA node arguments are ignored.

Finally, note that NVIDIA GPU drivers do not support suspending and resuming jobs. Attempting to suspend an application that makes use of GPU resources may result in unpredictable results including abnormal program termination.

8.4 Resynchronizing ALPS and the SDB Command After Manually Changing the SDB

Manual changes to node attributes and status can be reflected in ALPS by using the `apmgr resync` command. The `apmgr resync` command requests ALPS to reevaluate the configuration and attribute information and update its information. For example, after making manual changes to the SDB using the `xtprocadmin -e` or `xtprocadmin --noevent` command, use the `apmgr resync` command so that ALPS becomes aware of the changes.

8.5 Identifying Reserved Resources

The `apstat -r` command displays the batch job ID in the `From` field; for executables launched interactively, `apstat` displays `aprun` in the `From` field:

```
% apstat -r
  ResId     ApId  From          Arch   PEs N d Memory State
A  140  2497406 batch:741789     XT   512 - -   1333 conf,claim
   141  2497405 batch:741790     XT   768 24 1   1333 NID list,conf,claim
A  141  2497407 batch:741790     XT   768 - -   1333 conf,claim
```

The `apstat -A` *apid* command filters information by application IDs. You can include multiple application IDs, but it must be a space-separated list of IDs. For example:

```
% apstat -avv -A 3848874
Total (filtered) placed applications: 1
Placed  Apid ResId     User   PEs Nodes    Age   State Command
     3848874  1620 crayuser   512    22   0h07m  run   dnsp3+pat

Application detail
Ap[0]: apid 3848874, pagg 0x2907, resId 1620, user crayuser,
       gid 1037, account 0, time 0, normal
  Reservation flags = 0x2001
  Created at Tue Jul 12 14:20:08 2011
  Originator: aprun on NID 8, pid 6369
  Number of commands 1, control network fanout 32
  Network: pTag 131, cookie 0xfb860000, NTTgran/entries 1/22, hugePageSz 2M
  Cmd[0]: dnsp3+pat -n 512, 1365MB, XT, nodes 22
  Placement list entries: 512
  Placement list: 6-7,11,20-21,24-25,36-39,56-59,69-71,88-91
```

The `apstat -R` *resid* command filters information about reservation IDs. You can include multiple reservation IDs, but it must be a space-separated list of IDs. For example:

```
% apstat -rvv -R 1620
 ResId     ApId From           Arch   PEs N d Memory State
  1620  3848874 aprun            XT   512 0 1   1365 atomic,conf,claim

Reservation detail for resid 1620
Res[1]: apid 3848874, pagg 0, resId 619, user crayuser,
       gid 1037, account 8944, time 0, normal
  Batch System ID = 1971375
  Created at Tue Jul 12 14:20:08 2011
  Number of commands 1, control network fanout 32
  Cmd[0]: dnsp3+pat -n 512, 1365MB, XT, nodes 22
  Reservation list entries: 512
  Reservation list: 6-7,11,20-21,24-25,36-39,56-59,69-71,88-91
```

8.6 Terminating a Batch Job

To terminate a batch job, use the job ID from the `apstat -r` display.

8.7 Setting a Compute Node to Batch or Interactive Mode

To set a node to be either batch or interactive mode, use the `xtprocadmin` command to set the `alloc_mode` column of the SDB `processor` table. Then execute the `apmgr resync` command so that ALPS becomes aware of the changes.

Example 103. Retrieving node allocation status

The `apstat -n` command displays the application placement status of nodes that have a state of `UP` and their allocation mode (`B` for batch or `I` for interactive) in the State column.

Note: The `apstat` utility does not have dynamic run-time information about an application, so an `apstat` display does not imply anything about the running state of an application. An `apstat` display indicates statically that an application was placed and that the `aprun` claim against the reserved resources has not yet been released.

```
% apstat -n
  NID Arch State HW Rv Pl  PgSz     Avl    Conf  Placed  PEs Apids
   20   XT UP  B 12  -  -    4K 3072000       0       0    0
   21   XT UP  B 12  -  -    4K 3072000       0       0    0
   22   XT UP  B 12  -  -    4K 3072000       0       0    0
   23   XT UP  B 12  -  -    4K 3072000       0       0    0
<snip>
   63   XT UP  B 12 12 12    4K 3072000 3072000 1572864   12 221180
   64   XT UP  B 12 12 12    4K 3072000 3072000 1572864   12 221180
   65   XT UP  B 12 12 12    4K 3072000 3072000 1572864   12 221180
   66   XT UP  B 12 12 12    4K 3072000 3072000 1572864   12 221182
<snip>
Compute node summary
   arch config     up    use   held  avail   down
     XT    744    744     46     12    686      0
```

8.8 Manually Starting and Stopping ALPS Daemons on Service Nodes

ALPS is automatically loaded and started when CNL is booted on compute nodes.

You can manually start and stop the ALPS daemons on the service nodes as shown in the following procedures.

Procedure 68. Starting and stopping ALPS daemons on a specific service node

1. To start the ALPS daemons on a specific service node, log on to that service node as `root` and type the `/etc/init.d/alps start` command; for example, to start the ALPS daemons on the boot node:

   ```
   boot:~ # /etc/init.d/alps start
   ```

2. To stop the ALPS daemons on a specific service node, log on to that service node as `root` and type the `/etc/init.d/alps stop` command; for example, to stop the ALPS daemons on the boot node:

   ```
   boot:~ # /etc/init.d/alps stop
   ```

Procedure 69. Restarting ALPS daemon on a specific service node

- To restart the ALPS daemon on a specific service node, log on to the service node as `root` and type the `/etc/init.d/alps restart` command; for example, to restart the ALPS daemons on the boot node:

  ```
  boot:~ # /etc/init.d/alps restart
  ```

  The `/etc/init.d/alps restart` command stops and then starts the ALPS daemons on the node.

8.9 Manually Cleaning ALPS and PBS or TORQUE and Moab After Downed Login Node

If a login node goes down and will not be rebooted, job reservations associated with jobs deleted with `qdel` may not be released by ALPS. In this case, the `apstat -r` command lists the reservations as state `pendCancel` and leaves the jobs orphaned. Use the following procedure to manually clean up ALPS and the workload manager.

Procedure 70. Manually cleaning up ALPS and TORQUE and Moab or PBS after a login node goes down

1. Verify that the batch job still appears in the `qstat` output.

```
crayadm@smw:~> qstat -as 106728.sdb

sdb:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- -
-----
106728.sdb      root     workq    qsub.scrip   6231   1   1    --    --  R
00:00
   Job run at Thu Dec 03 at 14:31 on (login1:ncpus=1)
```

2. Purge job from the workload manager and verify that it was purged. On the SDB node, type:

   ```
   sdb:~ # qdel -W force 106728.sdb
   ```

3. Verify that the job no longer exists.

   ```
   sdb:~ # qstat -as 106728.sdb
   qstat: Unknown Job Id 106728.sdb
   sdb:~ #
   ```

4. Restart `apsched` on the SDB node:

   ```
   sdb:~# /etc/init.d/alps restart
   ```

5. Use `apmgr` to cancel the reservation that still exists in ALPS.

```
sdb:~ # apstat -r | grep 106728
 ResId     ApId From           Arch   PEs N d Memory State
     5  2949806 batch:106728     XT     1 0 1    500 conf

sdb:~ # apmgr cancel 5
```

6. Use `apstat` to verify that the reservation no longer exists.

```
sdb:~ # apstat -r | grep 106728
sdb:~ #
```

8.10 Verifying that ALPS is Communicating with Cray System Compute Nodes

Executing the following `aprun` command on a login node will return a list of host names of the Cray system compute nodes used to execute the last program.

Example 104. Verifying that ALPS is communicating with Cray system compute nodes

```
crayadm@login:~> cd /tmp
crayadm@login:/tmp> aprun -b -n 16 -N 1 /bin/cat /proc/sys/kernel/hostname
```

8.11 ALPS and Node Health Monitoring Interaction

The Node Health Checker (NHC) is invoked automatically upon the termination of an application or the end of a batch system reservation. Cleanup is therefore a two-stage process:

- Application cleanup is performed following an application exit.
- Reservation-level cleanup is performed following the termination of a batch system reservation.

System resources cannot be freed for reallocation until both the application cleanup and reservation cleanup have completed successfully.

8.11.1 Application Cleanup

During normal operations, an application exit is considered *orderly* when all of the application processes on a compute node have exited completely. ALPS gathers and consolidates all of this local exit information from each of the compute nodes within the application list; the exit information is sent to `aprun` over the ALPS application-specific TCP fan-out tree control network. When all of the exit information from all of the nodes is received by `aprun`, the `aprun` utility forwards the compiled information to `apsys` just before `aprun` itself exits.

Once all exit information has been received from the compute nodes, the application exit is considered orderly. An orderly exit does not necessarily mean that the application completed successfully, merely that exit information about the application was received by `aprun` and forwarded to `apsys`. The `apsys` utility then sends an exit message to `apsched`, which begins reservation-level cleanup.

An *unorderly* exit means that the node exit information has not been received by `apsys` before `aprun` exits. A typical example of an unorderly exit is when a `SIGKILL` signal is sent to `aprun` by the batch system after the application's wall time limit is exceeded. Since no exit information is available to `apsys` in an unorderly exit, `apsys` does not know the true state of the application processes on the compute nodes. Therefore, ALPS must perform application cleanup on each of the assigned compute nodes before it is safe to begin reservation cleanup.

Application cleanup begins with ALPS contacting each assigned compute node and sending a `SIGKILL` signal to any remaining application processes. Node health monitoring checks compute node conditions and marks a compute node `admindown` if it detects a problem. ALPS cannot free the node for reallocation until all of the application processes have exited or node health monitoring has marked the affected compute nodes either `admindown` or `suspect`. Until that time, the application will continue to appear in `apstat` displays.

8.11.1.1 aprun Actions

The `aprun` command is the ALPS application launch command on login nodes and the SDB node. The `aprun` command has a persistent TCP connection to a local `apsys`, as well as a persistent TCP connection to an `apinit` daemon child on the first compute node with in the assigned placement list, but not to an `apinit` on each assigned compute node.

After receiving a placement list from `apsched`, `aprun` writes information into the `syslog`, as in the example below.

```
May 18 10:38:16 nid00256 aprun[22477]: apid=1985825, Starting, user=10320,
batch_id=2325008, cmd_line="aprun -n 1 -b /tmp/hostname.xx ",
num_nodes=1, node_list=384

May 18 10:38:16 nid00256 aprun[22477]: apid=1985825, Error, user=10320,
batch_id=2325008, [NID 00384] 2010-05-18 10:38:15 Apid 1985825: cannot
execute: exit(107) exec failed

May 18 10:38:17 nid00256 apsys[22480]: apid=1985825, Finishing, user=10320,
batch_id=2325008
```

In a typical case of an orderly exit, `aprun` receives application exit information over the connection from that `apinit`. `aprun` then forwards the exit information over the connection to `apsys`. The ordering of application exit signals and exit codes is arbitrary. The `aprun` command displays any nonzero application exit information and uses the application exit information to determine its own exit code:

```
Application 284004 exit signals: Terminated
```

In the case of an unorderly exit, aprun exits without receiving application exit information. When aprun exits, its TCP connections are closed. The socket closes trigger application cleanup activity by both apinit and apsys.

An unorderly exit may occur for a variety of reasons, for example:

- The batch system sends a SIGKILL signal to aprun due to the application wall time expiring
- The apkill or kill commands are used to send a SIGKILL signal to aprun
- The aprun command receives a fatal message from apinit due to some fatal error during launch or at other points during the application lifetime, causing aprun to write the message to stderr and exit
- The aprun command receives a fatal read, write or unexpected close error on the TCP socket it uses to communicate with apinit

8.11.1.2 apinit Actions

The apinit daemon is the ALPS privileged daemon that launches and manages applications on compute nodes. For each application, the apinit daemon forks a child apshepherd process. Within ps displays, the child apshepherd processes retain the name "apinit".

The per-application TCP fan-out control tree has aprun as the root. Each compute node apshepherd within this control tree has a parent controller and may have a set of controlling nodes. Whenever a parent controller socket connection closes, the local apshepherd attempts to kill any application processes still executing and then will exit. This socket closing process results in a ripple effect through the fan-out control tree, resulting in automatic application tear down.

Whenever the aprun TCP connection to the apshepherd on the first compute node within the placement list closes, the tear-down process begins. During an application orderly exit, the exit information is sent to aprun, followed by the aprun closure of the socket connection, resulting in the exit of the apshepherd. The apshepherd exit causes its controlling socket connections to close as well. Each of those apshepherds will exit, and the application specific fan-out tree shuts down.

When the aprun TCP socket closure is not expected and the application processes are still executing, the apshepherd will send a SIGKILL signal to each local application process and then exit. There can be local delays in kernel delivery of the SIGKILL signal to the application processes due to application I/O activity. The application process will process the SIGKILL signal after the I/O completes. The apinit daemon is then responsible to monitor any remaining application processes.

This kill and exit process ripples throughout the control tree. However, if any compute node within the control tree is unresponsive, the ripple effect will stop for any compute nodes beyond that branch portion of the tree. In response to this situation, ALPS must take action independent of the shutdown of the control tree to ensure all of the application processes have exited or that compute nodes are marked either `admindown` or `suspect` by node health monitoring. The `apsys` daemon is involved in invoking the independent action.

8.11.1.3 apsys Actions

The `apsys` daemon is a local privileged ALPS daemon that runs on each login node and the SDB node. When contacted by `aprun`, the `apsys` daemon forks a child agent process to handle that specific local `aprun`. The `apsys` agent provides a privileged communication path between `aprun` and `apsched` for placement and exit information exchanges. The `apsys` agent name remains "`apsys`" within `ps` displays.

During an orderly application exit, the `apsys` agent receives exit information from `aprun` and forwards that information to `apsched`. However, during an unorderly exit, when the `aprun` socket connection closes prior to receipt of exit information, the `apsys` agent is responsible to start application cleanup on the assigned compute nodes.

To begin application cleanup, the `apsys` agent invokes cleanup and the `apsys` agent blocks until cleanup completes.

8.11.1.4 Application Cleanup Actions

Cleanup uses a tree-based overlay network formed using the `apinit` daemons on compute nodes associated with an unorderly exit to deliver a `SIGKILL` signal to application processes and to query for application presence. The overlay network is separate from the ALPS launch fan-out tree. All compute nodes that have a lingering application presence and all compute nodes with an unknown application presence status are gathered and used to inform the cleanup algorithm when to complete.

In the `apsys` log file, compute nodes that have a lingering application presence are reported in a `Match` list. Compute nodes with an unknown application presence status are reported in an `Unreached` list. The following example indicates that `apid 227061` remains resident on only one compute node (node 20), and that application presence status information has been received from all compute nodes:

```
14:00:21: Target Nodes: Match list portion for apid 227061 (1/1): 20
14:00:21: Target Nodes: Unreached list portion for apid 227061 (0/0):
```

After cleanup completes, or after every iteration of cleanup starting with the third iteration, the `xtcleanup_after` script is invoked.

8.11.2 Reservation-level Cleanup

Every time an application is launched on a set of compute nodes, ALPS creates a reservation tracker. When a reservation is terminated, either explicitly by the workload manager, implicitly upon orderly exit of the application, or via the `apmgr cancel` command, the information maintained by the reservation tracker is used to perform reservation-level cleanup.

Reservation-level cleanup is a two-step process, and largely transparent to the user. In the first step, the reservation is placed into a pending cancel state, regardless of the number of claims associated with the reservation or how the reservation was created. The `apsched` daemon then determines whether there are claims associated with the reservation, and if there are none, `apsched` instructs `apsys` to proceed with reservation cleanup. If there are claims associated with the reservation, then `apsys` performs application cleanup and keeps the reservation in pending cancel state until all application processes exit or the node in question is marked `admindown`.

In the second step, every node associated with the reservation and with an `up` status is targeted for reservation removal. The `apsys` agent then uses the `alpscleanup` library to remove the reservation. When this task is complete, `apsys` communicates to `apsched` that the reservation has been cleaned up, after which `apsched` removes the reservation from the ALPS shared state and sends confirmation back to `apsys`. The node is then returned to the pool of system resources available for new reservations.

8.11.3 Node Health Checker Actions

The Node Health Checker (NHC) is automatically invoked by ALPS upon the termination of an application. ALPS passes a list of nodes associated with the terminated application to NHC. NHC performs specified tests, which are specified in the NHC configuration file, to determine if compute nodes allocated to the application are healthy enough to support running subsequent applications. In particular, NHC attempts to reboot any nodes marked `admindown` or `suspect`. NHC removes any nodes incapable of running an application from the resource pool.

NHC verifies that the Application Level Placement Scheduler (ALPS) acknowledges a change that NHC has made to a node's state. If ALPS does not acknowledge a change, then NHC recognizes this disagreement between itself and ALPS. NHC then changes the node's state to `admindown` state and exits.

For an overview of NHC, see the `intro_NHC`(8) man page. For additional information about configuring node health checker, see Configuring Node Health Checker (NHC) on page 160.

8.11.4 Verifying Cleanup

There are a number of circumstances that can delay completion of application cleanup after an unorderly exit. This delay is often detected through `apstat` displays that still show the application and the resource reservation for that application.

As described in previous sections, check the various log files to understand what activity has taken place for a specific application.

- Check the `/var/opt/cray/alps/log/apsys`*YYYYMMDD* log files for that *apid*; verify cleanup has been invoked.
- Check the applicable node health monitoring log file (`/var/log/xtcheckhealth_log`) for that *apid*.
- Check the SMW `/var/opt/cray/log/`*sessionid*`/console-`*YYMMDDHHMM* log file for that *apid*.

Using Comprehensive System Accounting [9]

Comprehensive System Accounting (CSA) is open-source software that includes changes to the Linux kernel so that the CSA can collect more types of system resource usage data than under standard Fourth Berkeley Software Distribution (BSD) process accounting. CSA software also contains interfaces for the Linux process aggregates (`paggs`) and jobs software packages. The CSA software package includes accounting utilities that perform standard types of system accounting processing on the CSA generated accounting files. CSA, with Cray modifications, provides:

- Project accounting capabilities, which provide a way to charge computer system resources to specific projects
- An interface with various other job management systems in use at Cray sites
- A data management system for collecting and reporting accounting data
- An interface that you use to create the project account and user account databases, and to later modify them, as needed
- An interface that allows the project database to use customer–supplied user, account, and project information that resides on a separate Lightweight Directory Access Protocol (LDAP) server
- An interface with the ALPS application management systems so that application accounting records that include application start, termination, and placement information can be entered into the system accounting database

Specific third-party software releases are required for batch system compatibility with CSA on Cray systems. For more information, access the **3rd Party Batch SW** link on the CrayPort website at http://www.crayport.cray.com.

Complete features and capabilities of CSA are described in the `csa`(8) and `intro_csa`(8) man pages. The accounting utilities provided for administrative use are: `csanodeacct`, `csaperiod`, and `csarun`. The related man pages are accessible by using the `man` command.

Note: CSA runs **only** on login nodes and compute nodes. The SMW, boot node, SDB node, Lustre MDS nodes, and Lustre OSS nodes do not support CSA.

9.1 Interacting with Batch Entry Systems or the PAM `job` Module

Jobs are created on the system using either a batch job entry system (when such a system is used to launch jobs) or by the PAM `job` module for interactive sessions.

Note: You must be running TORQUE snapshot (release) `2.4.0-snap.20080925140` or later to take advantage of CSA support for the Cray platform.

You must run PBS Professional 9.2 or later to take advantage of CSA support for the Cray platform.

Compute node project accounting for applications submitted through workload managers (for example, PBS Professional) depends on the ability of the workload manager to obtain and propagate the project ID to ALPS at job submission time. If the workload manager does not support the ability to obtain and propagate the project ID to ALPS at job submission, the project ID must be set by using the `account` command prior to issuing an ALPS `aprun` command. Otherwise, project ID information will not be included in any CSA accounting records for the job.

9.2 CSA Configuration File Values

The CSA configuration file, `csa.conf`, is included with the Cray Linux Environment (CLE) software release package. This file contains default settings for several configuration parameters you must change to tailor CSA to your individual site configuration. On Cray systems `csa.conf` is located on the shared root in `/etc/opt/cray/csa/csa.conf` for login nodes and on the SMW in `/opt/xt-images/templates/default/etc/opt/cray/csa/csa.conf` for compute nodes.

Note: The two copies of this file **must** be identical with the exception of the `NODE_PROCESS_ACCOUNT` parameter.

Each Cray system that runs CNL has its own unique hardware configuration, including the number of nodes on the system, the physical location of the nodes, and a unique file system configuration. For this reason, the default `csa.conf` files can only be used as a template. A new version of the CNL compute node image must be created after editing `csa.conf` in order to implement the changes.

The parameters shown in the following table are used to define the accounting file system configuration and the node configuration for your system. You must change the settings of these parameters so that they conform to your system configuration.

Table 8. CSA Parameters That Must Be Specific to Your System

Parameter	Description
CSA_START	Defines whether CSA is enabled (on) or disabled (off). By default, CSA is disabled.
ACCT_SIO_NODES	Declares the number of account file system mount points. There must be at least one account file system mount point. The maximum number of mount points is 10. Multiple mount points are allowed so that the individual node accounting files can be distributed across more than one file system in order to provide better scaling for large system configurations. Use the df command to display the possible file system mount points. The actual maximum number of ACCT_SIO_NODES that may be specified is limited by the number of file systems available on your system.
ACCT_FILE_SYSTEM_00 ... ACCT_FILE_SYSTEM_*nn*	Must be one entry for each declared file system mount point. Numbering must begin with 00, and numbers must be consecutive. For example, if you have specified ACCT_SIO_NODES 1, you will only define ACCT_FILE_SYSTEM_00. If you have specified ACCT_SIO_NODES 2, you will also need to define ACCT_FILE_SYSTEM_01.
_lus_nid00023_csa_XT	The default file system mount point. It must be changed to correspond to a file system that exists on your system. There is one of these entries for each ACCT_FILE_SYSTEM declared. **Note:** The program that parses the configuration file does not allow any special characters, other than the underscore character (_) in configuration names. Therefore, in the file system paths used in the mount point description, each forward slash character (/) character must be represented by an underscore (_) character. This also means that an account file system mount point cannot have a _ character in the pathname.
SYSTEM_CSA_PATH	Defines the pathname on the common file system where CSA establishes its working directories for generating accounting reports. This parameter is only used on the service node image. It is not used on the compute nodes.
NODE_PROCESS_ACCOUNT	Defines whether all process account records written on a node will be written to the common file system, or whether the process account records for each application will be combined into a single application summary record that represents the total execution of the application on a node. This parameter may be set differently on the shared root and compute node images.

For other parameters in csa.conf, default settings should be acceptable.

9.3 Configuring CSA

CSA is disabled by default. When CSA is enabled, all system accounting, including service node accounting, is performed by CSA. Therefore, there is no need to have BSD process accounting enabled on service nodes. To enable CSA, set `CSA_START` to `on` in `csa.conf`.

Note: You must include the CSA RPM in your CNL boot image. If you set values in the `shell_bootimage.sh` script, make sure to edit the same values in `CLEinstall.conf` so that any new features remain enabled after the next CLE update or upgrade.

Perform the procedures in this section, in order, to correctly set up CSA.

9.3.1 Obtaining File System and Node Information

Procedure 71. Obtaining file system and node information

1. From a login node, enter the `df` command to determine which file systems are available for writing CSA accounting data.

```
login:~ > df

rootfs               173031424 158929920   5311488  97% /
initramdevs            8268844        76   8268768   1% /dev
10.131.255.254:/rr/current
                     173031424 158929920   5311488  97% /
10.131.255.254:/rr/current//.shared/node/8/etc
                     173031424 158929920   5311488  97% /etc
10.131.255.254:/snv   48070496  13872768  31755840  31% /var
10.131.255.254:/snv   48070496  13872768  31755840  31% /var
none                   8268844        12   8268832   1% /var/lock
none                   8268844       940   8267904   1% /var/run
none                   8268844         0   8268844   0% /var/tmp
tmpfs                  8268844        12   8268832   1% /tmp
ufs:/ufs              38457344  26436608  10067968  73% /ufs
ufs:/ostest           20169728  10874880   8269824  57% /ostest
23@gni:/lus_system     215354196400 60192583820 144222067004  30% /lus/nid00011
30@gni:/ib54ex       114821632416   5588944 108983420416   1% /lus/nid00064
```

2. Determine and record the file system information you want to use for CSA.

   The files systems of interest for saving accounting data are those two systems whose mount points are `/lus/nid00011` and `/lus/nid00064`, respectively. Record this information for later use.

3. Determine the hardware node configuration on your system.

Run the `xtprocadmin` command to get a complete list of nodes.

```
login:~ > xtprocadmin

   NID    (HEX)    NODENAME     TYPE    STATUS        MODE PSLOTS FREE
     0      0x0  c0-0c0s0n0  service        up       batch      4    0
     3      0x3  c0-0c0s0n3  service        up       batch      4    0
     4      0x4  c0-0c0s1n0  service        up       batch      4    4
     7      0x7  c0-0c0s1n3  service        up       batch      4    4
...
   475    0x1db  c3-0c2s6n3  compute        up       batch      4    4
   476    0x1dc  c3-0c2s7n0  compute        up       batch      4    4
   477    0x1dd  c3-0c2s7n1  compute        up       batch      4    4
   478    0x1de  c3-0c2s7n2  compute        up       batch      4    4
   479    0x1df  c3-0c2s7n3  compute        up       batch      4    4
```

For this example system, you want to choose a set of nodes that will have their accounting files written to `/lus/nid00011` and another set of nodes that will have their accounting files written to `/lus/nid00064`. You also need to make sure there is no overlap between the two sets of nodes.

9.3.2 Editing the `csa.conf` File

After you have the file system mount point and node configuration information for your system, you are ready to edit the `csa.conf` file. On Cray systems this file is located on the shared root at `/etc/opt/cray/csa/csa.conf` for login nodes and on the SMW at `/opt/xt-images/templates/default/etc/opt/cray/csa/csa.conf` for compute nodes.

Note: You must use `xtopview` to edit the shared root image of `csa.conf` file on the boot node. You can use any text editor to edit the compute node image of `csa.conf` file on the SMW. If the `/opt/xt-images/templates/default/etc/opt/cray/csa/csa.conf` files does not exist on the SMW, you may copy the file from the shared root.

Procedure 72. Editing CSA parameters for the example system

1. Enable CSA by setting `CSA_START` to `on`.
2. Set the number for the `ACCT_SIO_NODES` parameter.

   From Procedure 71 on page 300, you know that both `/lus/nid00011` and `/lus/nid000064` will be used to host individual node accounting files. The number of file systems (in this case two) to be used to contain accounting files is the value for the `ACCT_SIO_NODES` parameter. Since this example shows using `/lus/nid00011` and `/lus/nid00064` to contain accounting files, set `ACCT_SIO_NODES` to `2`:

   ```
   ACCT_SIO_NODES  2
   ```

3. Declare a file system mount point for each SIO node specified.

   Note: The program that parses the configuration file does not allow any special characters, other than the underscore character (_) in configuration names. Therefore, in the file system paths used in the mount point description, each forward slash character (/) character must be represented by an underscore (_) character. This also means that an account file system mount point cannot have a _ character in the pathname.

   The `df` command from the previous procedure showed a mount point on `/lus/nid00011` and another one on `/lus/nid00064`, these are the two mount points that need to be declared. Just because there are multiple mount points, however, does not mean that you need to use them. You may choose to have all accounting files written to a single file system. Since in this example you are configuring two mount points, you must specify `ACCT_FILE_SYSTEM_00` and `ACCT_FILE_SYSTEM_01` parameters:

   ```
   ACCT_FILE_SYSTEM_00  _lus_nid00011
   ACCT_FILE_SYSTEM_01  _lus_nid00064
   ```

4. Determine the node range values for the account system mount point parameters.

   All accounting file directories have `csa` as the first element of the path name, following the mount point. The next element in the path name after `csa` describes the node type. For Cray node types, the next element of the path name is `XT`.

   For Cray systems, the CSA software uses the node name, otherwise known as the *cname*, when creating pathnames for accounting files. For example, node name `c1-0c2s7n3` has a pathname of `cab1/row0/chassis2/slot7/mcomp3`. This path name is appended to applicable accounting system mount point name in order to create a full path name for the accounting file.

   The `xtprocadmin` command output from the previous procedure shows that the system has 4 cabinets, `c0-c3`. One simple way to configure the accounting file systems so that the files are divided fairly evenly between the two file systems in this example would be to specify that cabinet `0` and cabinet `1` have their data written to `/lus/nid00011`, and cabinet `2` and cabinet `3` have their data written to `/lus/nid00064`.

   Using the pathname conventions described above, and the node name data from Procedure 71 on page 300, you can define the file system mount point parameters:

   ```
   _lus_nid00011_csa_XT   c0-0c0s0n0--c1-0c2s7n3
   _lus_nid00064_csa_XT   c2-0c0s0n0--c3-0c2s7n3
   ```

5. Define the `SYSTEM_CSA_PATH` parameter.

   The `SYSTEM_CSA_PATH` parameter describes the file pathname for the system wide `csa` directories that are used for CSA work areas, and for containing the system-wide `pacct` file. The system-wide `pacct` file contains the merged

contents of the individual node `pacct` files. Since the file pathname for the `SYSTEM_CSA_PATH` is not used as an input to the configuration file parser, the file path name is allowed to contain the `/` character.

Usually the `SYSTEM_CSA_PATH` parameter uses an account file system mount point as its base directory, however, this is not required. The `SYSTEM_CSA_PATH` parameter is only used on the login node where CSA file processing is performed. It is not necessary to set this parameter in the compute node image of `csa.conf` on the SMW, but setting it there does not cause any problems.

For this example, use the `/lus/nid00011` mount point for the CSA work areas:

```
SYSTEM_CSA_PATH     /lus/nid00011/csa
```

6. Define the `NODE_PROCESS_ACCOUNT` parameter.

   The `NODE_PROCESS_ACCOUNT` parameter defines how much detailed accounting data is to be collected, processed, and saved from the nodes on the system. This parameter may be set differently in `/opt/xt-images/templates/default/etc/opt/cray/csa/csa.conf` on the SMW for the compute node image than in `/etc/opt/cray/csa/csa.conf` in the shared root file system for login nodes.

   To understand the usefulness of this parameter, it may be helpful to know how CSA accounting records are handled on Cray systems. When ALPS launches an application to the compute nodes on a Cray system, CSA process accounting occurs on each compute node. All CSA job and process accounting records for each compute node are written to an in-memory file system on the node, and the records remain there until the application terminates. When the application terminates, ALPS notifies the CSA software on each compute node to process the accounting data for that node. The `NODE_PROCESS_ACCOUNT` parameter allows CSA to make a decision whether to write all of the individual process accounting records for each compute node to the common file system, or to read the individual process accounting records and combine them into a single application summary record that represents the total resources used by the application on the compute node. By choosing to have application summary records, the amount of data transferred from each compute node to the common file system may be substantially reduced. In doing so, the amount of internal system network traffic and the amount of data moved from compute nodes to disk can be decreased. Also, the total amount of CSA accounting data that must be processed later for creating usage reports, and the amount of CSA data to be permanently stored can be reduced.

   You may want to set `NODE_PROCESS_ACCOUNT` off for compute nodes, and to set it on for service nodes. This provides more accounting process detail on

the login nodes where such information may be more useful. Therefore, this single parameter may be set differently on the shared root image than it is set on the compute node image of `csa.conf` on the SMW.

To use this split configuration, specify the following:

```
# Shared root version of /etc/opt/cray/csa/csa.conf:
NODE_PROCESS_ACCOUNT    ON

# Compute node image (on SMW) of /opt/xt-images/templates/default/etc/opt/cray/csa/csa.conf:
NODE_PROCESS_ACCOUNT    OFF
```

7. Change the parameter that defines the group name used for setting the ownership and group on accounting files. This parameter is named `CHGRP` and defaults to:

   ```
   CHGRP           csaacct
   ```

   If you use a different group name, change the parameter to match your system configuration.

9.3.3 Editing Other System Configuration Files

You also must make configuration changes to other system files. Use the `xtopview` command on the boot node to make the changes. For detailed information about using `xtopview`, see Managing System Configuration with the `xtopview` Tool on page 133 or the `xtopview`(8) man page.

- Add the `csaacct` user name to `/etc/passwd` on the shared root.

  ```
  csaacct:*:391:391:CSA:/var/lib/csa:/sbin/nologin
  ```

- Add the `csaacct` group name to `/etc/group` on the shared root.

  ```
  csaacct:!:391:
  ```

- Update the shadow password file to reflect the changes you have made:

  ```
  /usr/sbin/pwconv
  ```

- Add the `csaacct` group name to `/etc/group` on the CNL image.

  Note: The `csaacct` group and *gid* must be the same on the shared root and CNL image.

- Create additional PAM entries in `/etc/pam.d/common-session` to enable CSA. For more information about creating PAM entries, see Setting Up Job Accounting on page 308.

9.3.4 Creating a CNL Image with CSA Enabled

After you have modified the compute node copy of `csa.conf` on the SMW, you must rebuild the compute node image. For more information about how to rebuild the compute node image, see Preparing a Service Node and Compute Node Boot Image on page 63.

You can edit the shared root version of `csa.conf` and install the new version using the `xtopview` command. For more information about editing the shared root image of `csa.conf` using the `xtopview` utility, see Managing System Configuration with the `xtopview` Tool on page 133 or the `xtopview`(8) man page.

9.3.5 Setting Up CSA Project Accounting

The project database allows your site to define project names and assign an account number to each project. Users can have a list of account numbers that they can use for charging computing resources. Each user has a default account number that is assigned at login time.

Note: If you do not want to use CSA project accounting, complete Procedure 74 on page 307 instead of Procedure 73 on page 305.

Procedure 73. Setting up CSA project accounting

The project database resides on the system SDB node as a MySQL database. To set up a CSA project accounting for your system, perform this procedure.

1. Establish the project database, `UserProject`, and define the project database tables on the System Data Base (SDB) server:

```
sdb:~ # mysql -u root -h sdb -p < /opt/cray/projdb/default/sql/create_UserProject.sql
```

2. Grant administrative access privileges to the project database:

```
sdb:~ # mysql -u root -h sdb -p < /opt/cray/projdb/default/sql/create_accounts.sql
```

3. Use the `xtopview` command from the boot node to create and edit the `/etc/opt/cray/projdb/projects` file on the shared root so that it contains a list of valid account numbers and the associated project names.

   The `/etc/opt/cray/projdb/projects` file consists of a list of entries where each entry contains a project number followed by a project name. A colon character separates the project number from the project name. A project number and an account number are the same thing. The following example shows a simple project file:

```
0:root
100:sysadm
101:ProjectA
102:ProjectB
103:Big_Name_Project_that_is_insignificant_and_unimportant
1234567890:Big Name Project with Blanks in the Name
```

4. Use the `xtopview` command from the boot node to create and edit the `/etc/opt/cray/projdb/useracct` file on the shared root so that it contains a list of authorized users and the valid account numbers for each user.

   Each line of the user accounts file contains the login name of a user and list of accounts that are valid for that user. The first account number in the list is the user's default account. The default account number is assigned to the user at login time by the `pam_job` module. The user name is separated from the first account ID by a colon (`:`). Additional account numbers are separated by a comma (`,`).

   The following shows a simple user account file:

   ```
   root:0
   u1000:100
   u1001:101,103
   u1002:100,101
   u1003:100,103,1234567890
   ```

5. On the login node, edit the `~crayadm/.my.cnf` file in the home directory of the project database administrator so that it contains the following lines:

   ```
   [client]
   user=sys_mgmt
   password=sys_mgmt
   host=sdb
   ```

6. On the login node, change the permissions and owner on the `~crayadm/.my.cnf` file, as follows:

   ```
   login:/home/crayadm:~> chmod 600 ~crayadm/.my.cnf
   login:/home/crayadm:~> chown crayadm:crayadm ~crayadm/.my.cnf
   ```

7. If you are using customer–supplied user, account, and project information that resides on a separate LDAP server, use the `xtopview` command from the boot node to edit the `/etc/opt/cray/projdb/projdb.conf` project accounting configuration file so that it contains site-specific values for the parameters listed in Table 9.

Table 9. Project Accounting Parameters That Must Be Specific to Your System

Parameter	Description
PROJDBTYPE	If you are using customer–supplied user, account, and project information that resides on a separate LDAP server, change from MYSQL (default) to CUSTOM.
CUSTOM_VALIDATE	If you are using customer–supplied user, account, and project information that resides on a separate LDAP server, specify the full path name to the customer–supplied function that performs the necessary validation, for example /usr/local/sbin/validate_account. Input parameters to the validation function are position order dependent, as follows: user_name account_number

8. On a login node, run the projdb command with the -c option to create the project database. After the project database has been established, any users gaining access to the system through the job PAM module are assigned a default account ID at the time of system access.

```
login:/home/crayadm:~> projdb -c -p /etc/opt/cray/projdb/projects -u
/etc/opt/cray/projdb/useracct -v
```

Note: The project database package commands are installed in /opt/cray/projdb/default/bin, which must be in your PATH variable to access the commands.

9.3.5.1 Disabling Project Accounting

If you do not want to use project accounting on your site, either as provided by the MySQL database, or by a separate customer-supplied LDAP server, use the following procedure to disable project accounting.

Note: If you want to use CSA project accounting, complete Procedure 73 on page 305 instead of Procedure 74 on page 307.

Procedure 74. Disabling project accounting

1. In the /etc/opt/cray/projdb/projdb.conf file, set the PROJDBTYPE parameter to CUSTOM.
2. In the /etc/opt/cray/projdb/projdb.conf file, declare a CUSTOM_VALIDATE parameter and define it as /usr/local/sbin/validate_account.

3. As `root`, create the `/usr/local/sbin/validate_account` file with file permissions set to `755` and the following contents:

```
#!/bin/sh
echo 0
```

9.3.6 Setting Up Job Accounting

Note: You must include the `csa` RPM in your CNL boot image. To do this either set `CNL_csa=yes` in the `CLEinstall.conf` before the `CLEinstall` program is run or edit the `shell_bootimage_`*LABEL*`.sh` script and specify `CNL_CSA=y` prior to updating your CNL boot image. If you do set values in the `shell_bootimage.sh` script, make sure to edit the same values in `CLEinstall.conf` so that any new features remain enabled after the next CLE update or upgrade.

Procedure 75. Setting up CSA job accounting for non-CCM CNOS jobs

1. Cluster Compatibility Mode (CCM) does not support CSA accounting. However, the CNOS class **must** support CSA accounting for non-CCM jobs. To accomplish this, use the following configuration.

```
# xtopview  default/:/# vi /etc/opt/cray/ccm/ccm_mounts.local
/etc/pam.d/common-session-pc.ccm /etc/pam.d/common-session bind 0
default/:/# exit
```

2. In CNOS Class view:

```
common-session includes:
session optional pam_mkhomedir.so skel=/software/skel
session required pam_limits.so
session required pam_unix2.so
session optional pam_ldap.so
session optional pam_umask.s
session optional /opt/cray/job/default/lib64/security/pam_job.so
common-session.ccm does not include the pam_job entry:
session optional pam_mkhomedir.so skel=/software/skel
session required pam_limits.so
session required pam_unix2.so
session optional pam_ldap.so
```

For the procedure to disable CSA for the CNOS class view, see *Workload Management and Application Placement for the Cray Linux Environment* (S–2496).

For additional information about setting up job accounting on your system, read the `INSTALL` file that is included in the `job` RPM.

For more information about editing the shared root image of the `pam` configuration files using the `xtopview` utility, see Managing System Configuration with the `xtopview` Tool on page 133 or the `xtopview`(8) man page.

9.4 Creating Accounting `cron` Jobs

CSA depends on your system having a persistent `/var` file system for the shared root. For CSA to run successfully, you must establish the following `cron` jobs.

The normal order for the `cron` jobs is: `csanodeacct`, `csarun`, and then `csaperiod` (if necessary).

9.4.1 `csanodeacct` `cron` Job for Login Nodes

On Cray system compute nodes, when an application terminates, the Application Launch and Placement Scheduler (ALPS) initiates the CSA software that moves the local node accounting file records to a node-specific directory on the common file system (Lustre). On login nodes, this does not happen, and accounting records continue to accumulate indefinitely until the `csanodeacct` script is invoked to move the data to the common file system. Therefore, you need to periodically run a `cron` job on each login node to make sure that the local accounting files are moved as needed. This `cron` job must be owned by `root`.

Example 105. Running a `csanodeacct` `cron` job on each login node to move local accounting files

The following example shows moving accounting files from the local file system to the common file system on an hourly basis at 10 minutes before the hour. This `crontab` must be executed for each login node:

```
50 * * * * /opt/cray/csa/default/sbin/csanodeacct
```

9.4.2 `csarun` `cron` Job

You normally execute the `csarun` script at defined intervals to generate a set of system accounting reports.

Example 106. Executing the `csarun` script

To run `csarun` once per day at one minute before midnight, use a `crontab` entry of the following form:

```
59 23 * * * /opt/cray/csa/default/sbin/csarun
```

Note: This `crontab` must be executed from only **one** login node since it executes the `csanodemerg` script that merges all of the local node accounting files into a single system wide accounting file.

9.4.3 `csaperiod` `cron` Job

You can invoke the `csaperiod` script to run periodic accounting at different intervals than the regular system accounting interval.

Example 107. Running periodic accounting at different intervals than the regular system accounting interval

To run `csaperiod` once a week on Sunday at 5 minutes after midnight, use a `crontab` entry of the following form:

```
5 0 * * 0 /opt/cray/csa/default/sbin/csaperiod
```

Note: This `crontab` must be executed from only **one** login node since it executes the `csanodemerg` script that merges all of the local node accounting files into a single system-wide accounting file.

9.5 Enabling CSA

Using the `xtopview` command on the boot node is the only method to configure, enable, or disable services on the shared-root file system. You cannot configure, enable, or disable services on the login node itself. If your site has configured a `login` class for your system, invoke the following command sequence from the boot node as `root`:

```
boot# xtopview -x /etc/opt/cray/sdb/node_classes -c login
class/login:/# chkconfig job on
class/login:/# chkconfig csa on
class/login/# xtspec -c login /etc/init.d/job
class/login/# xtspec -c login /etc/init.d/csa
class/login:/# exit
```

On the subsequent system boot, this starts up the specified services on all nodes of the `login` class.

Note: If your site has not configured a `login` class, you must enable CSA for the individual login nodes using the `xtopview -n [`*nid#*`]` syntax rather than the `xtopview -c login` syntax shown. You must repeat the process for each login node. See the `xtopview`(8) man page for complete command option information.

9.6 Using LDAP with CSA

The `projdb` command and the `-l` option on the `account` command are not supported with customer-provided account validation.

The following Cray supplied library functions do not support this feature: `db_add_project`, `db_add_user`, `db_get_proj_acct`, `db_get_proj_name`, `db_get_user_accts`, `db_has_table`, `db_print_table`, `db_truncate_table`, and `db_validate_acct`.

For a description of the `/etc/opt/cray/projdb/projdb.conf` file and additional information on using a customer-supplied database, see the `projdb`(8) and `intro_csa`(8) man pages.

Dynamic Shared Objects and Cluster Compatibility Mode in CLE [10]

10.1 Configuring the Compute Node Root Runtime Environment (CNRTE) Using `CLEinstall`

Users can link and load dynamic shared objects in their applications by using the compute node root runtime environment (CNRTE) in the Cray Linux Environment (CLE). CLE includes software that enables compiling with dynamic libraries, using an alternate to the `initramfs` file system on the CNL compute nodes, called the compute node root. The compute node root is essentially the read-only DVS-projected shared root file system. This supports the ability to run a limited set of dynamically linked binaries on compute nodes.

The main benefit of this feature is expanded use of programs and libraries that require shared libraries on Linux cluster systems. If an independent software vendor (ISV) program ships with compiled binaries and dynamic libraries, you can also take advantage of this feature. Users are able to effectively reduce memory and executable footprint when shared objects, called multiple times, use the same segment of memory address space. Users can create applications that no longer need recompiling when libraries change.

Administrators enable this option at install time by modifying parameters in the `CLEinstall.conf` file.

For additional information, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) and *Workload Management and Application Placement for the Cray Linux Environment* (S–2496).

CNRTE is the framework used to allow compute node access to dynamic shared objects and libraries. Configuring and installing the compute node root runtime environment involves setting up the shared root as a DVS-projected file system. This process entails configuring DVS server nodes and updating the compute node boot images to enable them as clients.

To configure the compute node root runtime environment for CLE, do the following:

1. Determine which service or compute nodes will be the compute node root servers.

There are essentially two classes of nodes in a Cray system: service or compute. Service nodes have connectivity to external file systems and networks, access to the shared root of the boot node, and a full set of Linux services. Compute nodes have reduced services and a lightweight kernel to allow a maximized utilization of computational resources. Some services do not require external connectivity but are still desirable. There is also a practical limit to the number of available service nodes for each site. CLE allows you to run the service node image on a node otherwise considered a compute node to act as an internal DVS server of the Cray system shared root.

Note: Any compute nodes you choose here will no longer be a part of the available compute node pool. An allocation mode of `other` will be assigned to these compute nodes in the service database (SDB). These nodes will no longer belong to the group of batch and interactive nodes in the SDB and they will be unavailable to ALPS.

Caution: Do not place DVS servers on the same node as a Lustre (Object Storage, Metadata or Management) server. Doing so can cause load oversubscription on the node and reduce performance.

If the `/etc` files are specialized with a `cnos` class, the `cnos` class `/etc` files will be mounted on top of the projected shared root content on the compute nodes. This class specialization allows the compute nodes to have access to a different set of `/etc` files that exist on the DVS servers. Otherwise, the compute nodes will use the set of `/etc` files that are specific to their DVS server and that are contained in the shared root of the DVS server projects.

2. When editing the `CLEinstall.conf` file and running the `CLEinstall` program, modify the parameters specific to shared object support according to your site-specific configuration.

   When you set the following parameters in the `CLEinstall.conf` file, the `CLEinstall` program will automatically configure your system for the compute node root runtime environment.

   `DSL=`**`yes`** This variable enables dynamic shared objects and libraries for CLE. The default is `no`.

   Note: Setting this option to `yes` will automatically enable DVS.

   `DSL_nodes=`***`17 20`***

   The decimal NIDs of the nodes that will act as compute node root servers. These nodes can be a combination of service or compute nodes. Each NID is separated by a space.

`DSL_mountpoint=`***/dsl***

This path is the DVS mount point on the compute nodes; it is the projection of the shared root file system.

`DSL_attrcache_timeout=`***14400***

This value is the attribute cache time out for compute node root servers. The value represents the number of seconds before DVS attributes are considered invalid and they are retrieved from the server again.

3. Follow the appropriate procedures in *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444) to complete the installation.

The `/etc/opt/cray/cnrte/roots.conf` file contains site-specific values for custom root file systems. To specify a different pathname for `roots.conf` edit the configuration file `/etc/sysconfig/xt` and change the value for the variable, `CRAY_ROOTFS_CONF`. In the `roots.conf` file, the system default compute node root used is specified by the symbolic name `DEFAULT`. If no default value is specified, `/` will be assumed. In the following example segment of `roots.conf`, the default case uses `/dsl` as the reference root file system:

```
DEFAULT=/dsl
INITRAMFS=/
DSL=/dsl
```

Users can override the system default compute node root value by setting the `CRAY_ROOTFS` environment variable to a value from the `roots.conf` file. This changes the compute node root used for launching jobs. For example, to override the use of `/dsl` set `CRAY_ROOTFS` to `INITRAMFS`.

An administrator can modify the contents of this file to restrict user access. For example, if the administrator only wants to allow applications to launch using the compute node root, the `roots.conf` file would read like the following:

```
% cat /etc/opt/cray/cnrte/roots.conf
DEFAULT=/dsl
```

10.2 Configuring Cluster Compatibility Mode

A Cray XE series system is not a cluster but a massive parallel processing (MPP) computer. An MPP is simply one computer with many networked processors used for distributed computation, and, in the case of Cray XE architectures, a high-speed communications processor that facilitates optimal bandwidth and memory operations between those processors. When operating as an MPP machine, the Cray compute node kernel (Cray CNL) typically does not have a full set of the Linux services available that are used in cluster ISV applications.

Cluster Compatibility Mode (CCM) is a software solution that provides the services needed to run most cluster-based independent software vendor (ISV) applications out-of-the-box. CCM supports ISV applications running in four simultaneous cluster jobs on up to 256 CNL compute nodes per job instance. It is built on top of the Compute Node Root Runtime Environment (CNRTE), the infrastructure used to provide dynamic library support in Cray systems.

CCM is tightly coupled to the workload management system. It enables users to execute cluster applications alongside workload-managed jobs running in a traditional MPP batch or interactive queue. Essentially, CCM uses the batch system to logically designate part of the Cray system as an emulated cluster for the duration of the job as shown in Figure 5 and Figure 6.

Figure 5. Cray System Job Distribution Cross-section

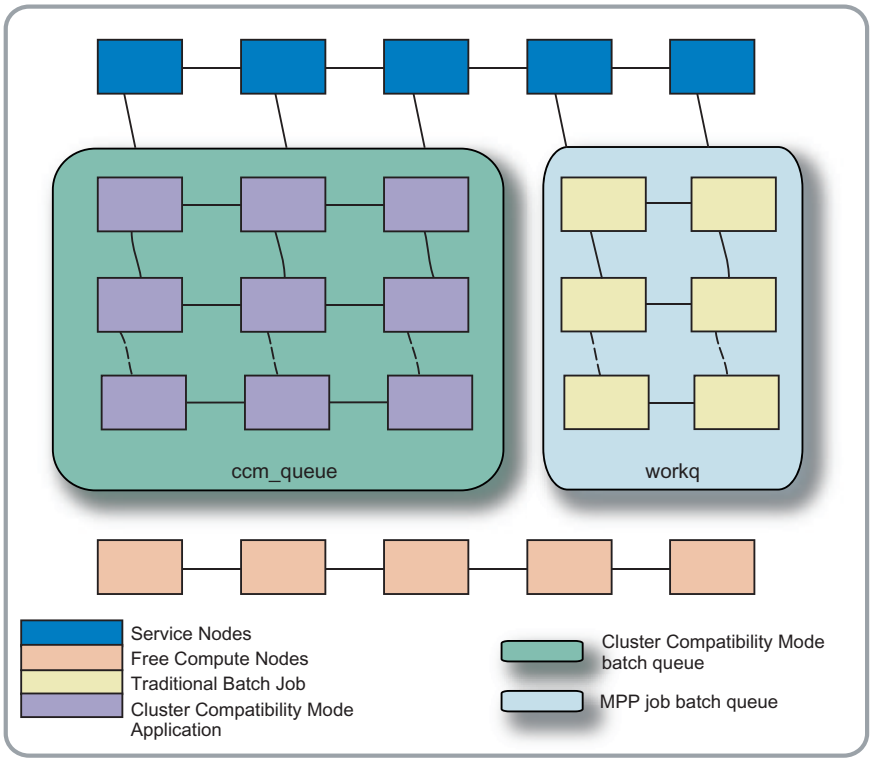
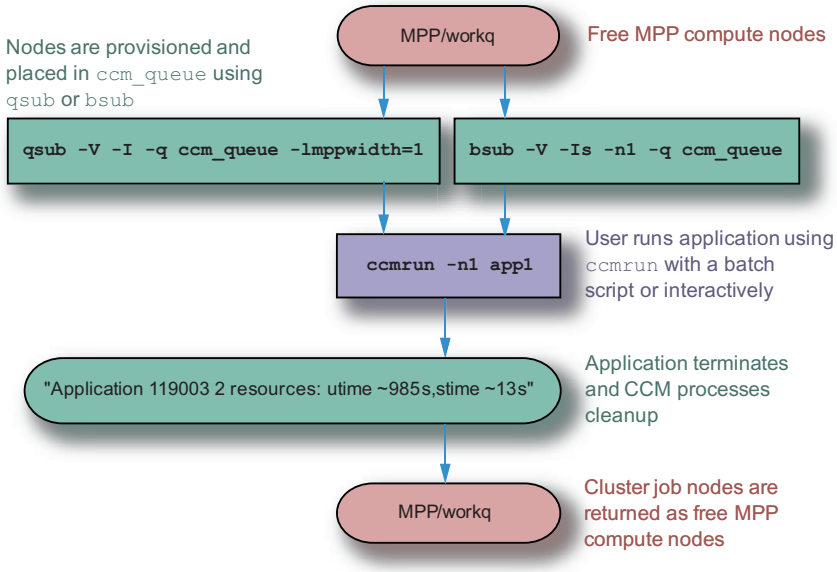


Figure 6. CCM Job Flow Diagram



10.2.1 Preconditions

- Dynamic library support is installed.
- (Optional) RSIP must be installed if you have applications that need access to a license server; see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).
- PBS 10.2RC2 (Emerald), LSF 8.0, or `Torque-2.4.1b1-snap.200908271407` or later versions are installed.

10.2.2 Configuration Options

The following are exclusively post-install options included in /etc/opt/cray/ccm/ccm.conf:

CCM_DEBUG=*no*

Setting this option to yes enables debug logging for CCM. These logs will be stored on the PBS MOM node in/var/log/crayccm. Cray recommends the site setting this option to yes.

CCM_RESOURCES="*ccm ccm2*"

This option indicates that the administrator has configured a custom application resource that can be allocated and used for a job. Users requesting a job will consume one of the pool of available resources listed here. The job submission is checked against the list provided when making a determination whether the job is a CCM targeted job.

Note: Only one of CCM_RESOURCES or CCM_QUEUES is required.

To configure yp, /etc/defaultdomain and /etc/yp.conf must be properly configured on the compute node specialized view. Cray recommends that you use the cnos class within xtopview to set up this specialized view.

Procedure 76. Using DVS to mount home directories on the compute nodes for CCM

For each DVS server node you have configured, mount the path to the user home directories. Typically, these will be provided from a location external to the Cray system.

1. Specialize and add a line to the /etc/fstab file on the DVS server by using xtopview in the node view. For example, if your DVS server is c0-0c0s2n3 (node 27 on a Cray XE system), type the following:

```
boot:~ # xtopview -m "mounting home dirs" -n 27
node/27:/ # xtspec -n 27 /etc/fstab
node/27:/ # vi /etc/fstab
nfs_home_server:/home        /home    nfs     tcp,rw  0 0
node/27:/ # exit
```

2. Log into each DVS server and mount the file system:

```
boot:~ # ssh nid00027
nid00027:~ # mount /home
nid00027:~ # exit
```

3. To allow the compute nodes to mount their DVS partitions, add an entry in the `/etc/fstab` file in the compute image for each DVS file system. For example:

```
smw:~ # vi /opt/xt-images/templates/default/etc/fstab
/home /home dvs path=/home,nodename=c0-0c0s2n3
```

4. For each DVS mount in the `/etc/fstab` file, create a mount point in the compute image.

```
smw:~ # mkdir -p /opt/xt-images/templates/default/home
```

5. Update the boot image to include these changes; follow the steps in Procedure 2 on page 64.

   Note: You can defer this step and update the boot image **once** before you finish booting the system.

Procedure 77. Modifying CCM and Platform-MPI system configurations

1. If you wish to enable additional features such as debugging and Linux NIS (Network Information Service) support, edit the CCM configuration file by using `xtopview` in the default view.

```
boot:~ # xtopview -m "configuring ccm.conf"
default/:/ # vi /etc/opt/cray/ccm/ccm.conf
```

   If you wish to configure additional CCM debugging, set `CCM_DEBUG=yes`.

   If you wish to enable NIS support, set `CCM_ENABLENIS=yes`.

2. (Optional) You may have a site configuration where the paths for the `qstat` command is not at a standard location. Change the values in the configuration file for `CRAY_QSTAT_PATH` and `CRAY_BATCH_VAR` accordingly for your site configuration.

3. Save and close `ccm.conf`.

4. Exit `xtopview`.

```
default/:/ # exit
boot:~ #
```

   Important: If your applications will use Platform-MPI (previously known as *HP-MPI*), Cray recommends that users populate their `~/.hpmpi.conf` (or `~/.pmpi.conf`) file with these values.

```
MPI_REMSH=ssh
MPIRUN_OPTIONS="-cpu_bind=MAP_CPU:0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\
22,23,24,25,26,27,28,29,30,31"
```

Procedure 78. Setting up files for the cnos class

The `cnos` compute nodes that have access to the shared root through CNRTE will have a specialized class of its own `/etc` files. Login files and all `/etc` files should be migrated to the `cnos` class in order for CCM to work.

1. Use `xtopview` to access the `cnos` class specialized files:

   ```
   boot:~#  xtopview -m "CCM cnos setup" -c cnos
   ```

   Note: If the SDB has not been started, use the `-x /etc/opt/cray/sdb/node_classes` option to specify node/class relationships.

2. To add a file or modify a file, edit the file and then specialize it for the `cnos` class.

   ```
   class/cnos:/# vi /etc/file
   class/cnos:/# xtspec -c cnos /etc/file
   ```

   Repeat the above steps for each new file that you want to add or modify for the compute nodes.

3. Exit `xtopview`.

   ```
   class/cnos:/# exit
   ```

   Note: You are prompted to type **c** and enter a brief comment describing the changes you made. To complete your comment, type **Ctrl-d** or a period on a line by itself. Do this each time you exit `xtopview` to log a record of revisions into a version control system.

Procedure 79. Linking the CCM prologue/epilogue scripts for use with PBS and Moab TORQUE on login nodes

Prerequisites: This procedure requires that you have already installed a workload management system such as PBS or Moab TORQUE.

Add a line to reference to append the CCM prologue and epilogue scripts to the end of the existing batch prologue and epilogue. The PBS batch prologue is configured on all PBS MOM nodes in `/var/spool/PBS/mom_priv/prologue.` The Moab TORQUE batch prologue is configured on all TORQUE MOM nodes in `/var/spool/torque/mom_priv/prologue.`

Note: This procedure assumes that you are using `/bin/bash` as your shell, but this can be modified appropriately for others.

1. Add the following lines to prologue:

   ```
   #!/bin/bash
   ccm_dir=/opt/cray/ccm/default/etc

   if [ -x $ccm_dir/cray-ccm-prologue ] ; then
      $ccm_dir/cray-ccm-prologue $1 $2 $3
   fi
   ```

2. Add the following lines to epilogue:

```
#!/bin/bash
ccm_dir=/opt/cray/ccm/default/etc

if [ -f $ccm_dir/cray-ccm-epilogue ] ; then
  $ccm_dir/cray-ccm-epilogue $1 $2 $3 $4 $5 $6 $7 $8 $9
fi
```

3. Set the executable bit for `prologue` and `epilogue` if not set:

```
system :/var/spool/PBS/mom_priv # chmod a+x prologue epilogue
```

4. Change the default batch time-out value. Cray recommends changing this to 120 seconds. This allows the system enough time to startup and shutdown all infrastructure on the nodes associated with the CCM job. To change the batch time out, append the following line to `/var/spool/PBS/mom_priv/config` or `/var/spool/torque/mom_priv/config`:

```
$prologalarm 120
```

Procedure 80. Using `qmgr` to create a general CCM queue and queues for separate ISV applications

1. Set up a general CCM queue by issuing the following `qmgr` commands on the PBS server node:

```
# module load pbs
# qmgr
Qmgr: create queue ccm_queue
Qmgr: set queue ccm_queue queue_type = Execution
Qmgr: set queue ccm_queue resources_max.mpparch = XT
Qmgr: set queue ccm_queue resources_min.mpparch = XT
Qmgr: set queue ccm_queue resources_min.mppwidth = 1
Qmgr: set queue ccm_queue resources_default.mpparch = XT
Qmgr: set queue ccm_queue resources_default.mppwidth = 1
Qmgr: set queue ccm_queue enabled = True
Qmgr: set queue ccm_queue started = True
Qmgr: exit
```

   For Moab TORQUE, add this command while creating the queue:

```
set server query_other_jobs = True
```

2. Repeat step 1 for additional application-specific queues, if desired.

Procedure 81. Configuring Platform LSF for use with CCM

Prerequisites: This procedure requires that you have already installed the Platform LSF workload management system.

1. Determine the path to the directory on your system that contains the files `lsb.queues` and `lsb.params`. This path is

`${LSF_TOP}/conf/lsbatch/${LSF_CLUSTER_NAME}/configdir`, where `LSF_TOP` and `LSF_CLUSTER_NAME` are themselves paths that were defined at install time.

Example 108. Location of queue configuration files

If

```
LSF_TOP=/opt/xt-lsfhpc
```

and

```
LSF_CLUSTER_NAME=nid00196
```

the full path to the directory containing the queue configuration files would be:

```
/opt/xt-lsfhpc/conf/lsbatch/nid00196/configdir.
```

2. Create a `ccm_queue` for Platform LSF. Refer to Platform documentation for information on managing LSF queues.

3. Enable `PRE_EXEC` and `POST_EXEC` scripts for the queue set up in Procedure 80 on page 319 by setting the following parameters in `lsb.queues`:

```
QUEUE_NAME = ccm_queue
PRE_EXEC = /opt/cray/ccm/default/etc/lsf_ccm_pre
POST_EXEC = /opt/cray/ccm/default/etc/lsf_ccm_post
LOCAL_MAX_PREEXEC_RETRY=1
DESCRIPTION=ccm_queue
```

   To enable LSF using an application profile rather than a queue, set the following in `lsb.applications`:

```
Begin Application
NAME=ccm
DESCRIPTION=Sets up an application profile for CCM
PRE_EXEC=/opt/cray/ccm/default/etc/lsf_ccm_pre
POST_EXEC=/opt/cray/ccm/default/etc/lsf_ccm_post
LOCAL_MAX_PREEXEC_RETRY=1
```

   For more information on the `lsb.applications` file, see the *Platform LSF Configuration Reference* manual.

4. In the file `lsb.params` set the `JOB_INCLUDE_POSTPROC` to ensure that the job reservation remains in a running state until execution of the `POST_EXEC` script completes and all necessary clean up has finished:

```
JOB_INCLUDE_POSTPROC=Y
```

5. On the boot node shared root file system, update `/etc/lsf.sudoers` using `xtopview`:

```
boot:~ # xtopview
default/:/ # vi /etc/lsf.sudoers
```

   Make the `root` user the `LSB_PRE_POST_EXEC_USER`:

```
LSB_PRE_POST_EXEC_USER=root
```

6. Exit the editor and change the default permissions for `/etc/lsf.sudoers` so that the batch system infrastructure can properly communicate with compute nodes:

   ```
   default/:/ # chmod 600 /etc/lsf.sudoer
   ```

7. Exit `xtopview`.

Once you have completed system configuration and started the system compute nodes, you should verify that write permissions are correct. You can accomplish this by using `touch` to create a dummy file within CCM:

```
% ccmrun touch foo
```

If `foo` is created in the user directory then the write permissions are set correctly.

Procedure 82. Creating custom resources with PBS

1. Edit the `/var/spool/PBS/server_priv/resourcedef` file on the SDB node.
2. Add the following line:

   ```
   ccm type=boolean
   ```

3. Invoke one of the following commands to submit your custom resource.

   - Batch:

     qsub -lmppwidth=*width* -lccm=1 *job_script*.pbs

   - Interactive:

     qsub -I -lmppwidth=*width* -lccm=1 ./*job_script*.pbs

Procedure 83. Creating custom resources with Moab

Moab custom resources are managed as generic global node resources. These can be configured in the `moab.cfg` file in the installed Moab spool directory (e.g., `/var/spool/moab/moab.cfg`.

1. Add the following entry to `moab.cfg` to allow up to 1024 `ccm` instances on the system at one time:

   ```
   NODECFG[GLOBAL] GRES=ccm:1024
   ```

2. To consume this resource at runtime, invoke the following command:

   qsub -I -lmppwidth=1 -lgres=ccm

   The result of this submit is that the following information is set for the job:

   ```
   Resource_List.gres = ccm
   ```

Using InfiniBand and OpenFabrics Interconnect Drivers [11]

InfiniBand (IB) and OpenFabrics remote direct memory access (RDMA) is supported on service nodes for Cray systems running the Cray Linux Environment (CLE) operating system.

No separate installation is required. The kernel-space libraries and drivers are built against Cray's kernel. OFED and InfiniBand RPMs are included in the CLE release and installed by default. However, OFED will not run on your Cray system until you configure the I/O nodes to use IB.

To configure IB and OFED, see the procedures provided in this chapter; to configure IB and OFED during installation or upgrade of your CLE software, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

11.1 InfiniBand and OFED Overview

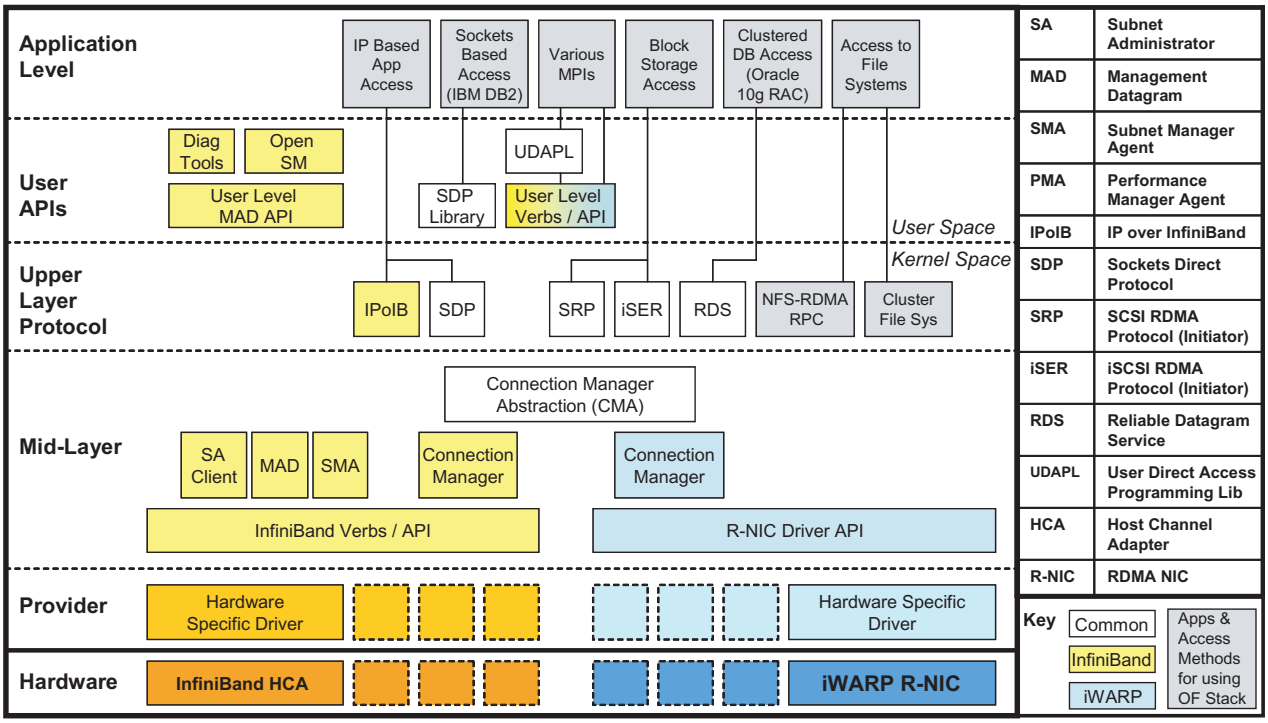
Cray supports InfiniBand as an I/O interconnect. IB enables efficient zero-copy, low-latency RDMA transfers between network peers. As a result, IB gives Cray systems the most efficient transfer mechanism from the high speed network (HSN) to external I/O devices.

CLE includes a subset of the OpenFabrics Enterprise Distribution (OFED) to support the use of InfiniBand on Cray I/O nodes. OFED is the software stack on the host that coordinates user-space and kernel-space access to the IB hardware. IB support is restricted to I/O service nodes that are equipped with PCI Express (PCIe) cards for network connectivity.

IB can be used on Lustre router nodes as a network interconnect between the Cray system and external Lustre servers.

The OFED software stack consists of many different components. These components can be categorized as kernel modules (drivers) and user/system libraries and utilities, commands and daemons for InfiniBand administration, configuration, and diagnostics; Cray maintains the kernel modules so that they are compatible with CLE.

Figure 7. The OFED Stack (source: OpenFabrics Alliance)



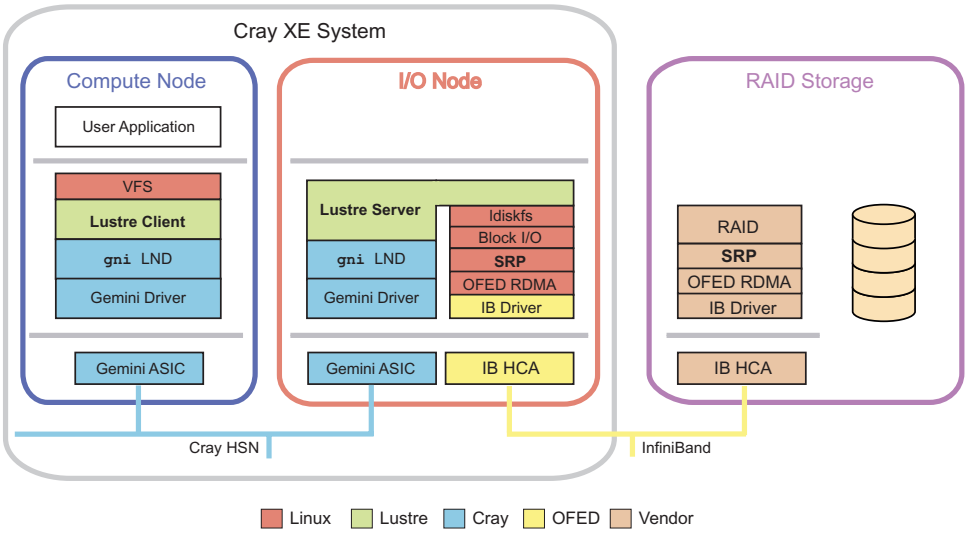
11.2 Using InfiniBand

InfiniBand is a payload-agnostic transport. It can move small messages or large blocks efficiently between network endpoints. The following examples demonstrate how Cray uses InfiniBand and the OFED stack to support block I/O, file I/O, and standard network inter-process communication.

11.2.1 Storage Area Networking

InfiniBand can transport block I/O requests to external storage targets. ANSI T10's SCSI RDMA Protocol (SRP) is currently the only SCSI-transporting protocol supported on Cray systems with InfiniBand. Figure 8 shows SRP on InfiniBand connecting the Cray to an external RAID array. The OFED stack is shown in the storage array for clarity; it is provided by your site-specific third party storage vendor.

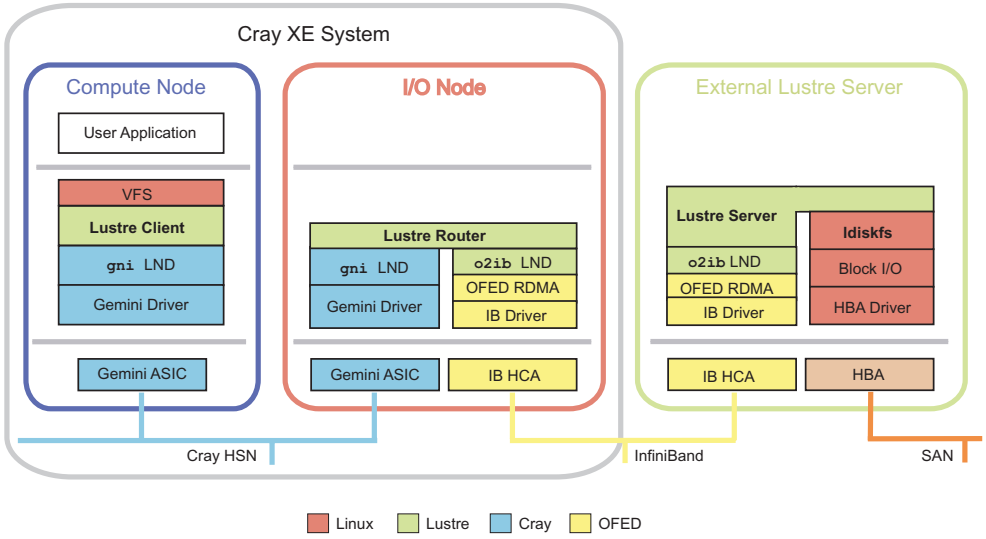
Figure 8. Cray System Connected to Storage Using SRP



11.2.2 Lustre Routing

Cray uses InfiniBand on the service nodes to connect Cray compute nodes to Lustre File System by Cray (CLFS) servers, as shown in Figure 9. In this configuration, the Cray service node is no longer a Lustre server. Instead, it runs a Lustre router provided by the LNET layer. The router moves LNET messages between the Cray HSN and the external IB network, which transports file-level I/O requests between the clients on the Cray HSN and the servers over the IB fabric. Please speak with your Cray service representative regarding an CLFS solution for your Cray system.

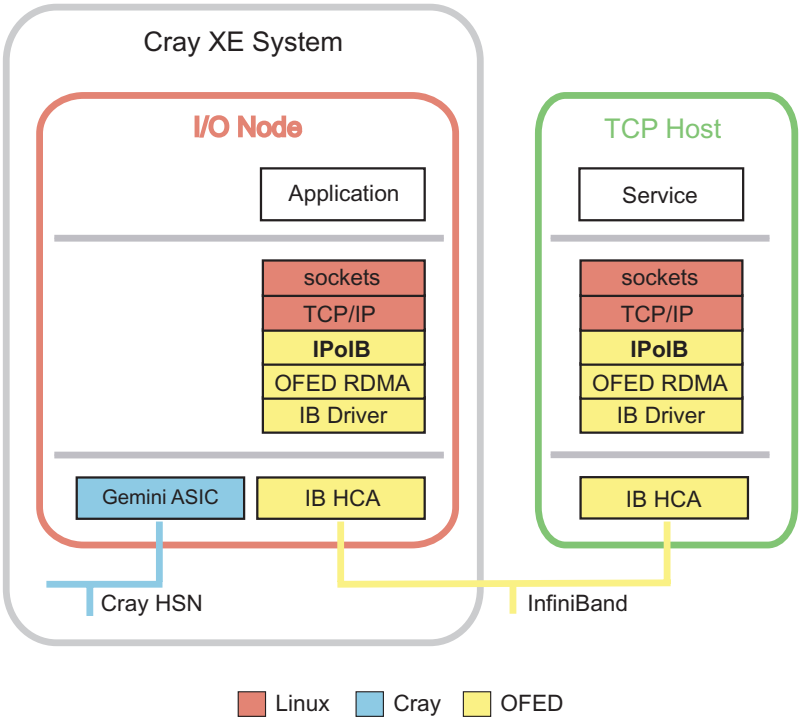
Figure 9. Cray Service Node Acting as an InfiniBand Lustre Router



11.2.3 IP Connectivity

InfiniBand can also carry socket-based inter-process traffic typical of commodity clusters and TCP/IP networking. InfiniBand supports the IP over IB (IPoIB) protocol. Since IB plugs-in below the socket interface, neither the application nor the service needs to be recompiled to communicate over an InfiniBand network. Both protocols are diagrammed on a service node in Figure 10.

Figure 10. Cray Service Node in IP over IB Configuration



11.3 Configuration

In addition to the OFED RDMA stack, Cray supports three upper layer protocols (ULPs) on its service nodes as shown in Table 10. Because all ULPs use the OFED stack, the InfiniBand Configuration must be followed for all IB service nodes.

Note: It is only necessary to configure the specific ULPs that you intend to use on the service node.

For example, a Lustre server with an IB direct-attached storage array uses the SCSI RDMA Protocol (SRP), not the LNET Router. On the other hand, if the Lustre servers are external to the Cray system, the service node uses the LNET router instead of SRP. IP over InfiniBand (IPoIB) is used to connect non-RDMA socket applications across the IB network.

Table 10. Upper Layer InfiniBand I/O Protocols for Cray Systems

Upper Layer Protocol	Purpose
IP over IB (IPoIB)	Provides IP connectivity between hosts over IB.
Lustre (OFED LND)	Base driver for Lustre over IB. On service nodes, this protocol enables efficient routing of LNET messages from Lustre clients on the Cray HSN to external IB-connected Lustre servers. The name of the LND is `o2iblnd`.
SCSI RDMA Protocol (SRP)	T10 standard for mapping SCSI over IB and other RDMA fabrics. Supported by DDN and LSI for their IB-based storage controllers.

11.4 InfiniBand Configuration

Procedure 84. Configuring InfiniBand on service nodes

InfiniBand includes the core OpenFabrics stack and a number of upper layer protocols (ULPs) that use this stack. Configure InfiniBand by modifying `/etc/sysconfig/infiniband` for each IB service node.

1. Use the `xtopview` command to access service nodes with IB HCAs.

   For example, if the service nodes with IB HCAs are part of a node class called `lnet`, type the following command:

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -c lnet
   ```

   Or

   Access each IB service node by specifying either a node ID or physical ID. For example, access node 27 by typing the following:

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -n 27
   ```

2. Specialize the `/etc/sysconfig/infiniband` file:

   ```
   node/27:/ # xtspec -n 27 /etc/sysconfig/infiniband
   ```

3. Add IB services to the service nodes by using standard Linux mechanisms, such as executing the `chkconfig` command while in the `xtopview` utility or executing `/etc/init.d/openibd start | stop | restart` (which starts or stops the InfiniBand services immediately). Use the `chkconfig` command to ensure that IB services are started at system boot.

   ```
   node/27:/ # chkconfig --force openibd on
   ```

4. While in the `xtopview` session, edit `/etc/sysconfig/infiniband` and make these changes.

   ```
   node/27:/ # vi /etc/sysconfig/infiniband
   ```

   a. By default, IB services do not start at system boot. Change the `ONBOOT` parameter to `yes` to enable IB services at boot.

      ```
      ONBOOT=yes
      ```

   b. By default at boot time, the Internet Protocol over InfiniBand (IPoIB) driver loads on all nodes where IB services are configured. Verify that the value for `IPOIB_LOAD` is set to `yes` to enable IPoIB services.

      ```
      IPOIB_LOAD=yes
      ```

      Important: LNET routers use IPoIB to select the paths that data will travel via RDMA.

   c. The SCSI RDMA Protocol (SRP) driver loads by default on all nodes where IB services are configured to load at boot time. If your Cray system needs SRP services, verify that the value for `SRP_LOAD` is set to `yes` to enable SRP.

      ```
      SRP_LOAD=yes
      ```

      Important: Direct-attached InfiniBand file systems require SRP; Lustre file systems external to the Cray system do not require SRP.

5. Exit `xtopview`.

   ```
   node/27:/ # exit
   boot:~ #
   ```

   Note: You are prompted to type **`c`** and enter a brief comment describing the changes you made. To complete your comment, type **`Ctrl-d`** or a period on a line by itself. Do this each time you exit `xtopview` to log a record of revisions into an RCS system.

6. Proper IPoIB operation requires additional configuration. See Procedure 86 on page 330.

11.5 Subnet Manager (OpenSM) Configuration

InfiniBand fabrics require at least one Subnet Manager (SM) operating on each IB subnet in order to activate its respective IB port connected to the fabric. This is one critical difference between IB fabrics and Ethernet, where simply connecting a cable to an Ethernet port is sufficient to get an active link. Managed IB switches typically include an SM and, therefore, do not require any additional configuration of any of the hosts. Unmanaged IB switches, which are considerably less expensive, do not include a SM and, thus, at least one host connected to the switch must act as a subnet manager. InfiniBand standards also support switchless (point-to-point) connections as long as an SM is installed. An example of this case is when a service node is connected to direct-attached storage through InfiniBand.

The OpenFabrics distribution includes OpenSM, an open-source IB subnet management and subnet administration utility. Either one of the following configuration steps is necessary if no other subnet manager is available on the IB fabric. The subnet manager RPMs are installed in the shared root by running `CLEinstall`. OpenSM can be started from the service node on a single port at boot time or manually from the command line to load multiple instances per host.

11.5.1 Starting OpenSM at Boot Time

Procedure 85. Starting a single instance of OpenSM on a service node at boot time

Note: This procedure assumes that the IB HCA is in node 8.

1. Use `xtopview` to access the service node that will host your instance of OpenSM.

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -n 8
   ```

2. Specialize `/etc/sysconfig/opensm` for the IB node.

   ```
   node/8:/ # xtspec -n 8 /etc/sysconfig/opensm
   ```

3. Edit `/etc/sysconfig/opensm` to have OpenSM start at boot time

   ```
   # To start OpenSM automatically set ONBOOT=yes
   ONBOOT=yes
   ```

4. Add IB services to the service nodes by using standard Linux mechanisms, such as executing the `chkconfig` command while in the `xtopview` utility or executing `/etc/init.d/opensmd start|stop|restart|status` (which starts or stops the OpenSM service immediately). The `chkconfig` command can be used to ensure that the OpenSM service is started at system boot.

   ```
   node/8:/ # /sbin/chkconfig --force opensmd on
   ```

11.6 Internet Protocol over InfiniBand (IPoIB) Configuration

Procedure 86. Configuring IP Over InfiniBand (IPoIB) on Cray systems

1. Use `xtopview` to access each service node with an IB HCA by specifying either a node ID or physical ID. For example, to access node 27, type the following:

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -n 27
   ```

2. Specialize the `/etc/sysconfig/network/ifcfg-ib0` file.

   ```
   node/27:/ # xtspec -n 27 /etc/sysconfig/network/ifcfg-ib0
   ```

3. Modify the site-specific `/etc/sysconfig/network/ifcfg-ib0` file on each service node with an IB HCA.

   ```
   node/27:/ # vi /etc/sysconfig/network/ifcfg-ib0
   ```

   For example, to use static IP address, *172.16.0.1*, change the `BOOTPROTO` line in the file.

   ```
   BOOTPROTO='static'
   ```

   Add the following lines to the file.

   ```
   IPADDR='172.16.0.1'
   NETMASK='255.128.0.0'
   ```

   To configure the interface at system boot, change the `STARTMODE` line in the file.

   ```
   STARTMODE='onboot'
   ```

4. (Optional) If you would like to configure IPoIB for another IB interface connected to this node, repeat step 2 and step 3 for `/etc/sysconfig/network/ifcfg-ib`*n*.

   Note: For LNET traffic, each IB interface should be assigned a unique IP address from the subnet that it will operate on. For TCP/IP traffic, multiple IB interfaces on a node must be assigned unique IP addresses from different subnets.

11.7 Configuring SCSI RDMA Protocol (SRP) on Cray Systems

Procedure 87. Configuring and enabling SRP on Cray Systems

1. Use the `xtopview` command to access service nodes with IB HCAs.

   For example, if the service nodes with IB HCAs are part of a node class called `ib`, type the following command:

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -c ib
   ```

2. Edit `/etc/sysconfig/infiniband`

   ```
   ib/:/ # vi /etc/sysconfig/infiniband
   ```

and change the value of SRP_DAEMON_ENABLE to yes:

```
SRP_DAEMON_ENABLE=yes
```

3. Edit srp_daemon.conf to increase the maximum sector size for SRP.

```
ib/:/ # vi /etc/srp_daemon.conf

a       max_sect=8192
```

4. Edit /etc/modprobe.conf.local to increase the maximum number of gather-scatter entries per SRP I/O transaction.

```
ib/:/ # vi /etc/modprobe.conf.local

options ib_srp srp_sg_tablesize=255
```

5. Exit from xtopview.

```
ib/:/ # exit
boot:~ #
```

11.8 Lustre Networking (LNET) Router

LNET is the Lustre networking layer. LNET isolates the file system code from the Lustre Networking Drivers (LNDs), which provide an interface to the underlying network transport. For more information on Lustre networking please see the Lustre documentation at http://wiki.whamcloud.com/display/PUB/Documentation.

Although LNET is automatically loaded with the Lustre servers and clients, it can be launched by itself to create a standalone router between networks instantiated by LNDs. LNET routing is most efficient when the underlying transports are capable of remote direct memory access (RDMA). Cray Lustre currently supports LNDs for a number of RDMA transports, including gnilnd, which is used for Cray systems using the Gemini or Aries interconnects, and o2iblnd, which is used for the OpenFabrics InfiniBand stack. Cray builds and distributes the OFED LND (o2iblnd) and gnilnd as part of its Lustre distribution.

Note: LNET routing is also available over GigE and 10GigE networks with socklnd, although this configuration does not support RDMA.

Routing Lustre involves configuring router nodes to route traffic between Lustre servers and Lustre clients which are on different networks. Routing Lustre requires that three types of nodes be configured: the router, the client, and the InfiniBand server. LNET assigns node IDs to each node on the network. While gnilnd uses node IDs (for example, *nnn*@gni) to enumerate ports on the Cray side, o2iblnd uses InfiniBand IP addresses (for example, *nn.nn.nnn.nnn*@o2ib) for hosts. As a result, IPoIB must be configured on each IB port. See Internet Protocol over InfiniBand (IPoIB) Configuration on page 330 for more information. For the rest of this discussion, assume that LNET routers are being created on two Cray service

nodes, both of which have a single IB port connected to a switched InfiniBand fabric. The network configuration is shown in Internet Protocol over InfiniBand (IPoIB) Configuration on page 330.

Table 11. LNET Network Address Configuration for Cray Systems

`gni` Address	Network Component	InfiniBand Address
27	Router 1	10.10.10.28
31	Router 2	10.10.10.32
10.128.0.255	IP Subnet	10.10.10.255
255.255.255.0	Subnet Mask	255.255.255.0

11.8.1 Configuring the LNET Router

Procedure 88. Configuring the LNET routers

The following description covers the configuration of the LNET router nodes.

1. Use `xtopview` to access the default view of the shared root.

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes
   ```

2. Create your `/etc/init.d/lnet` router controller (RC) script. An RC script is necessary to start LNET in the absence of any Lustre file services. A sample RC script is available in Appendix H, Sample LNET Router Controller Script on page 421.

   Note: Cray does not provide an RC script with its release packages. You must verify that this script will work for your configuration or contact your Cray service representative for more information.

   a. Create the `/etc/init.d/lnet` file in the default view.

      ```
      default/:/ # vi /etc/init.d/lnet
      ```

   b. Copy the sample script into your new file and write it.

3. Use `chkconfig` to enable LNET since there are no mounts or Lustre server activity to load the LNET module implicitly.

   ```
   default/:/ # /sbin/chkconfig lnet on
   ```

4. Add the following LNET directives to the Cray shared root in `/etc/modprobe.conf.local`.

```
options lnet ip2nets="gni0 10.128.0.*; o2ib 10.10.10.*"
options lnet routes="gni0 10.10.10.[28,32]@o2ib;  o2ib [27,31]@gni0"
```

   Note: Larger system configuration LNET directives may exceed the 1024 character limit of `modprobe.conf` entries. Procedure 89 on page 333 allows administrators to source `ip2nets` and `routes` information from files to work around this limitation.

   `o2ib` is the LNET name for the OFED LND and `gni` is the LNET name for the Cray LND. Here `ip2nets` is used instead of `networks` because it provides for an identical `modprobe.conf` across all Lustre clients in the Cray system.

   The `ip2nets` directive tells LNET to load both LNDs and associates each LND with an IP subnet. It overrides any previous `networks` directive (for example, `lnet networks=gni`). On service nodes without an IB adapter, the `o2ib` LND does not load because there are no ports with the IP subnet used defined in `ip2nets`.

   Note: Each Cray system sharing an external Lustre file system must have a unique `gni` identifier for the LNET options listed in this step. In this case, the Cray system is using `gni0`. Other systems would use other numbers to identify their LNET networks (such as `gni1`, `gni2`, and so on).

5. Cray recommends enabling these options to improve network resiliency. Edit `/etc/modprobe.conf.local` on the Cray shared root to include:

```
options ko2iblnd timeout=100 peer_timeout=130
options ko2iblnd credits=2048 ntx=2048
options ko2iblnd peer_credits=126 concurrent_sends=63 peer_buffer_credits=128
options kgnilnd credits=2048 peer_health=1

options lnet check_routers_before_use=1
options lnet dead_router_check_interval=60
options lnet live_router_check_interval=60
options lnet router_ping_timeout=50
options lnet large_router_buffers=1024 small_router_buffers=16384
```

   Note: You will need to configure these values for your specific system size.

6. Exit from `xtopview`.

   You are prompted to add a comment about the operations you have performed. Enter **c**, and then enter a brief comment about the changes you made to the file.

Procedure 89. Specifying service node LNET `routes` and `ip2nets` directives with files

1. Enter `xtopview` and create the following files (if they do not already exist):

```
boot:~ # xtopview
default/:/ # touch /etc/lnet_ip2nets.conf
default/:/ # touch /etc/lnet_routes.conf
```

2. Exit `xtopview`.

   ```
   default/:/ # exit
   ```

3. Enter the correct view of `xtopview` for the service node or node class that you wish to modify `modprobe` directives for and specialize the two files you created in step 1.

   ```
   boot:~ # xtopview -c login
   class/login:/ # xtspec /etc/lnet_ip2nets.conf
   class/login:/ # xtspec /etc/lnet_routes.conf
   ```

4. Add the following lines to `/etc/lnet_ip2nets.conf` (substitute your site-specific values here).

   *gni0 10.128.0.**
   *o2ib 10.10.10.**

5. Add the following lines to `/etc/lnet_routes.conf` (substitute your site-specific values here).

   gni0 10.10.10.[28,32]@o2ib
   o2ib [27,31]@gni0

6. Add the following LNET directives to the Cray shared root in `/etc/modprobe.conf.local`.

   ```
   options lnet ip2nets="/etc/lnet_ip2nets.conf"
   options lnet routes="/etc/lnet_routes.conf"
   ```

7. Exit `xtopview`.

   ```
   class/login:/ # exit
   ```

Procedure 90. Manually controlling LNET routers

- If `/etc/init.d/lnet` is not provided, send the following commands to each LNET router node to control them manually:
  - Startup:

    ```
    modprobe lnet
    lctl net up
    ```

  - Shutdown:

    ```
    lctl net down
    lustre_rmmod
    ```

11.8.2 Configuring Routes for the Lustre Server

Procedure 91. Configuring the InfiniBand Lustre Server

The Lustre servers must be configured with proper routes to allow them to reach the Cray client nodes on the gni network. If there are multiple Cray systems involved, each Cray must use a unique `gni` network identifier (for example, `gni0`, `gni1`). The server `routes` configuration maps each `gni` network to the corresponding Cray router nodes. Add an `lnet routes` directive for each Cray system. Perform these steps on the remote host:

1. Edit `/etc/modprobe.conf` on the remote host to include the route to the LNET network.

   ```
   options lnet ip2nets="o2ib"
   options lnet routes="gni0 10.10.10.[28,32]@o2ib"
   ```

   If there are two Cray systems accessing the file system exported by these hosts, then both Cray systems must be included in the `lnet routes` directive.

   ```
   options lnet routes="gni0 10.10.10.[28,32]@o2ib; \
   gni1 10.10.10.[71,72,73,74]@o2ib"
   ```

   In this example, there are two Cray systems: `gni0` with two router nodes and `gni1` with four.

2. Add the following options to `/etc/modprobe.conf` to improve network resiliency :

   ```
   options ko2iblnd timeout=100 peer_timeout=0 keepalive=30
   options ko2iblnd credits=2048 ntx=2048
   options ko2iblnd peer_credits=126 concurrent_sends=63

   options lnet avoid_asym_router_failure=1
   options lnet dead_router_check_interval=60
   options lnet live_router_check_interval=60
   options lnet router_ping_timeout=50
   options lnet check_routers_before_use=1
   ```

   Because Lustre is running on the external host, there is no need to start LNET explicitly.

11.8.3 Configuring the LNET Compute Node Clients

Procedure 92. Configuring the LNET Compute Node Clients

Since compute nodes are running the Lustre client, they do not need explicit commands to start LNET. There is, however, additional configuration required for compute nodes to be able to access the remote Lustre servers via the LNET routers. These changes are made to `/etc/modprobe.conf` for the compute node image used in booting the system.

1. Edit `/etc/modprobe.conf` for the compute node boot image. The `lnet`

`ip2nets` directive identifies the LND. If there is more than one Cray system sharing the file system, then this identifier (`gni`) must be unique for each Cray system.

```
options lnet ip2nets="gni0"
options lnet routes="o2ib [27,31]@gni0"
```

2. Add the following options to `/etc/modprobe.conf` to improve network resiliency :

```
options lnet avoid_asym_router_failure=1
options lnet dead_router_check_interval=60
options lnet live_router_check_interval=60
options lnet router_ping_timeout=50
options lnet check_routers_before_use=1
```

3. (Optional) For InfiniBand-connected LNET clients (such as Cray Development and Login (CDL) nodes), add the following options to `/etc/modprobe.conf`:

```
options ko2iblnd timeout=100 peer_timeout=0 keepalive=30
options ko2iblnd credits=2048 ntx=2048
options ko2iblnd peer_credits=126 concurrent_sends=63
```

4. Modify `/etc/fstab` in the compute node boot image to identify the external server. The file system is described by the LNET node ID of the MGS server (and its failover partner if Lustre failover is configured)

```
10.10.10.1@o2ib:10.10.10.2@o2ib: /boss1 /mnt/boss1 lustre rw,flock,lazystatfs
```

Here, the `fstab` mount option `rw` gives read/write access to the client node. The `flock` option is to allow Lustre's client node to have exclusive access to the file lock, and the `lazystatfs` option prevents command hangs (such as `df`) if one or more OSTs are unavailable.

In this example, the Lustre file system with the `fsname "boss1"` is provided by the Lustre management server (MGS) on the InfiniBand fabric at IP address `10.10.10.1` (with `10.10.10.2` being the failover partner). Because both routers have access to this subnet, the Lustre client performs a round-robin with its requests to the routers.

5. Update the boot image to include these changes; follow the steps in Procedure 2 on page 64.

Important: Accessing any externally supplied Lustre file system requires that both the file server hosts and the LNET routers be up and available before the clients attempt to mount the file system. Boot time scripts in the compute node image take care of reading `fstab` and running the necessary `mount` commands. In production, this is the only opportunity to run Lustre `mount` commands because kernel modules get deleted at the end of the boot process.

11.9 Configuring Fine-grained Routing with `clcvt`

The `clcvt` command, available on the boot node and the SMW, aids in the configuration of LNET fine-grained routing (FGR). FGR is a routing scheme that aims to group sets of Lustre servers on the file system storage array with LNET routers on the Cray system. This grouping maximizes file system performance on larger systems by using a router-to-server ratio where the relative bandwidth is roughly equal on both sides. FGR also minimizes the number of LNET network hops (hop count) and file system network congestion by sending traffic to particular Lustre servers over dedicated network lanes instead of the default round-robin configuration.

The `clcvt` command takes as input several file-system-specific files and generates LNET kernel module configuration information that can be used to configure the servers, routers, and clients for that file system. The utility can also create cable maps in HTML, CSV, and human-readable formats and validate cable connection on installed systems. For more information, such as available options and actions for `clcvt`, see the `clcvt`(8) man page.

11.9.1 Prerequisite Files

The `clcvt` command requires several prerequisite files in order to compute the `ip2nets` and `routes` information for your specific configuration. Before `clcvt` can be executed for the first time, these files must be placed in an empty directory on the boot node or SMW, depending on where you run `clcvt`.

Deciding how to assign which routers to which OSSs, what FGR ratios to use, which interface on which router to use for a LNET group, and router placement are all things that can vary greatly from site to site. LNET configuration is determined as the system is ordered and configured; see your Cray representative for your site-specific values.

`info.`*file-system-identifier*

A file with global file system information for the *cluster-name* server machine and each client system that will access it.

client-system`.hosts`

A file that maps the client system (such as the Cray mainframe) IP addresses to unique host names, such as the boot node `/etc/hosts` file. The *client-system* name must match one of the `clients` in the `info.`*file-system-identifier* file.

client-system`.ib`

A file that maps the client system LNET router InfiniBand IP addresses to system hardware `cnames`. The *client-system* name must match one of the `clients` in the `info.`*file-system-identifier* file. This file must be created by an administrator.

clustername`.ib`

A file that maps the Lustre server InfiniBand IP addresses to cluster (for example, Sonexion) host names. The *clustername* name must match the `clustername` in the `info.`*file-system-identifier* file. This file must be created by an administrator.

client-system`.rtrIm`

A file that contains `rtr -Im` command output (executed on the SMW) for the *client-system*.

11.9.1.1 `info.`*file-system-identifier*

`info.`*file-system-identifier* is a manually-created file that contains global file system information for the Lustre server machine and each client system that will access it. Based on the ratio of server to LNET routers in your configuration, the `[`*clustername*`]` section and each `[`*client-system*`]` section will define which servers and routers will belong to each InfiniBand subnet.

This file is of the form of a `ini` style file, and the possible keywords in the `[info]` section include `clustername`, `ssu_count`, and `clients`.

`clustername` defines the base name used for all file system servers. For a Sonexion file system, as in the example below, it might be something like `snxs11029n` and all server hostnames will be `snxs11029n`*NNN*, where *NNN* is a three digit number starting at `000` and `001` for the primary and secondary Cray Sonexion Management Servers (CSMS), `002` for the MGS, `003` for the MDS, `004` for the first OSS, and counting up from there for all remaining OSSs.

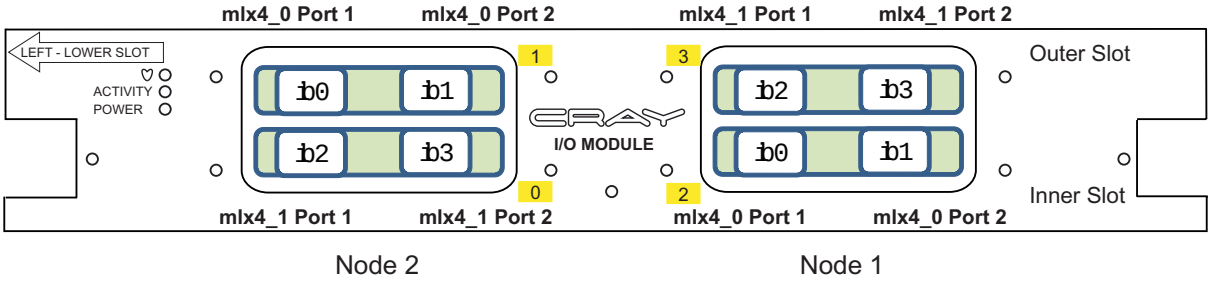
`ssu_count` defines how many SSUs make up a Sonexion file system. If this is missing, then this is not a Sonexion file system but an CLFS installation.

`clients` defines a comma-separated list of mainframe names that front-end this file system.

The `info.`*file-system-identifier* file also needs a `[`*client-system*`]` section for each client system listed in the `clients` line of the `[info]` section to describe the client systems and a `[`*clustername*`]` section to describe the Lustre server system. Each of these sections contain a literal `lnet_network_wildcard` in the format of *LNET-name*:*IP-wildcard* which instructs the LNET module to match a host's IP address to *IP-wildcard* and, if it matches, instantiate LNET *LNET-name* on them. Example 109 shows a sample `info.`*file-system-identifier* configuration file.

The hostname fields in the `[`*client-system*`]` section of this file are fully-qualified interface specifications of the form *hostname*`(ib`*n*`)`, where `(ib0)` is the assumed default if not specified. Cray XC30 systems can support multiple InfiniBand interfaces per router. Configure the second IB interface (see Procedure 86 on page 330) and append the interface names (`ib`*n*`)` to the `cname`s for the routers. See Example 110 for reference. You will also need to append these interface names to the *client-system*`.ib` file. InfiniBand port assignments are shown below in Figure 11.

Figure 11. Cray XC30 InfiniBand Port Assignment



Example 109. Sample `info.`*file-system-identifier* file: info.*snx11029*

```
# This section describes the size of this filesystem.
[info]
clustername = snx11029n
SSU_count = 6
clients = hera

[hera]
lnet_network_wildcard = gni1:10.128.*.*

# Because of our cabling assumptions and naming conventions, we only
# need to know which XIO nodes are assigned to which LNETs.  From that
# our tool can actually generate a "cable map" for the installation folks.
o2ib6000: c0-0c2s2n0, c0-0c2s2n2 ; MGS and MDS
o2ib6002: c1-0c0s7n0, c1-0c0s7n1, c1-0c0s7n2, c1-0c0s7n3 ; OSSs 2/4/6
o2ib6003: c3-0c1s5n0, c3-0c1s5n1, c3-0c1s5n2, c3-0c1s5n3 ; OSSs 3/5/7
o2ib6004: c3-0c1s0n0, c3-0c1s0n1, c3-0c1s0n2, c3-0c1s0n3 ; OSSs 8/10/12
o2ib6005: c3-0c2s4n0, c3-0c2s4n1, c3-0c2s4n2, c3-0c2s4n3 ; OSSs 9/11/13

[snx11029n]
lnet_network_wildcard = o2ib6:10.10.100.*

o2ib6000: snx11029n002, snx11029n003 ; MGS and MDS
o2ib6002: snx11029n004, snx11029n006, snx11029n008 ; OSSs 2/4/6
o2ib6003: snx11029n005, snx11029n007, snx11029n009 ; OSSs 3/5/7
o2ib6004: snx11029n010, snx11029n012, snx11029n014 ; OSSs 8/10/12
o2ib6005: snx11029n011, snx11029n013, snx11029n015 ; OSSs 9/11/13
```

Example 110. Sample `info.`*file-system-identifier* file using multiple IB interfaces per router

```
[info]
clustername = snx11014n
SSU_count = 4
clients = crystal

[crystal]
lnet_network_wildcard = gni0:10.128.*.*

o2ib1000: c2-0c0s1n1, c2-0c1s1n1 ; MGS and MDS
o2ib1002: c0-0c0s1n2(ib0), c3-0c0s0n1(ib0), c3-0c0s0n2(ib0) ; OSSs 4/6/8/10
o2ib1003: c0-0c0s1n2(ib2), c3-0c0s0n1(ib2), c3-0c0s0n2(ib2) ; OSSs 5/7/9/11

[snx11014n]
lnet_network_wildcard = o2ib:10.10.100.*

o2ib1000: snx11014n002, snx11014n003 ; MGS and MDS
o2ib1002: snx11014n004, snx11014n006, snx11014n008, snx11014n010 ; OSSs 4/6/8/10
o2ib1003: snx11014n005, snx11014n007, snx11014n009, snx11014n011 ; OSSs 5/7/9/11
```

11.9.1.2 *client-system*`.hosts`

For a typical Cray system, this file can be the `/etc/hosts` file taken from the boot node. Simply make a copy of the `/etc/hosts` file from the boot node and save it to a working directory where you will later run the `clcvt` command.

Example 111. Sample *client-system*.hosts file: *hera*.hosts

```
#
# hosts         This file describes a number of hostname-to-address
#               mappings for the TCP/IP subsystem.  It is mostly
#               used at boot time, when no name servers are running.
#               On small systems, this file can be used instead of a
#               "named" name server.
# Syntax:
#
# IP-Address  Full-Qualified-Hostname  Short-Hostname
#

127.0.0.1       localhost

# special IPv6 addresses
::1     ipv6-localhost  localhost       ipv6-loopback

fe00::0 ipv6-localnet

ff00::0 ipv6-mcastprefix
ff02::1 ipv6-allnodes
ff02::2 ipv6-allrouters
ff02::3 ipv6-allhosts
# Licenses
172.30.74.55    tic     tic.us.cray.com
172.30.74.56    tac     tac.us.cray.com
172.30.74.57    toe     toe.us.cray.com
172.30.74.206   cflls01 cflls01.us.cray.com
172.30.74.207   cflls02 cflls02.us.cray.com
172.30.74.208   cflls03 cflls03.us.cray.com
##LDAP Server Info
172.30.12.46    kingpin kingpin.us.cray.com     kingpin.cray.com
172.30.12.48    kingfish        kingfish.us.cray.com    kingfish.cray.com
##esLogin Info
172.30.48.62    kiyi    kiyi.us.cray.com        el-login0.us.cray.com
10.2.0.1        kiyi-eth1
##Networker server
#172.30.74.90   booboo  booboo.us.cray.com


10.3.1.1        smw
10.128.0.1      nid00000        c0-0c0s0n0      dvs-0
10.128.0.2      nid00001        c0-0c0s0n1      boot001 boot002
10.128.0.31     nid00030        c0-0c0s0n2      #old ddn6620_mds
10.128.0.32     nid00031        c0-0c0s0n3      hera-rsip2
10.128.0.3      nid00002        c0-0c0s1n0
10.128.0.4      nid00003        c0-0c0s1n1
10.128.0.29     nid00028        c0-0c0s1n2
10.128.0.30     nid00029        c0-0c0s1n3
10.128.0.5      nid00004        c0-0c0s2n0
10.128.0.6      nid00005        c0-0c0s2n1
10.128.0.27     nid00026        c0-0c0s2n2
10.128.0.28     nid00027        c0-0c0s2n3
10.128.0.7      nid00006        c0-0c0s3n0
10.128.0.8      nid00007        c0-0c0s3n1
10.128.0.25     nid00024        c0-0c0s3n2
10.128.0.26     nid00025        c0-0c0s3n3
```

```
10.128.0.9      nid00008        c0-0c0s4n0      login   login1  hera
10.128.0.10     nid00009        c0-0c0s4n1      sdb001  sdb002
10.128.0.23     nid00022        c0-0c0s4n2      hera-rsip  hera-rsip1
10.128.0.24     nid00023        c0-0c0s4n3      mds nid00023_mds
...
```

11.9.1.3 *client-system*`.ib`

The *client-system*`.ib` file contains client-system LNET router InfiniBand IP address to `cname` mapping information in a `/etc/hosts` style format. The hostname field in this file is a fully-qualified interface specification of the form *hostname*`(ib`*n*`)`, where `(ib0)` is the assumed default if not specified. Cray XC30 systems can support multiple InfiniBand interfaces per router; configure the second IB interface (see Procedure 86 on page 330) and append the interface names (`ib`*n*`)` to the `cname`s for the routers. The LNET router InfiniBand IP addresses should be within the same subnet as the Lustre servers (MGS/MDS/OSS); one possible address assignment scheme would be to use a contiguous set of IP addresses, with `ib0` and `ib2` on each node having adjacent addresses. See Example 113 for reference. You will also need to append these interface names to the `info.`*file-system-identifier* file. This file must be created by an administrator.

Example 112. Sample *client-system*`.ib` file: *hera*.ib

```
#
# This is the /etc/hosts-like file for Infiniband IP addresses
# on "hera".
#
10.10.100.101    c0-0c2s2n0
10.10.100.102    c0-0c2s2n2
10.10.100.103    c1-0c0s7n0
10.10.100.104    c1-0c0s7n1
10.10.100.105    c1-0c0s7n2
10.10.100.106    c1-0c0s7n3
10.10.100.107    c3-0c1s0n0
10.10.100.108    c3-0c1s0n1
10.10.100.109    c3-0c1s0n2
10.10.100.110    c3-0c1s0n3
10.10.100.111    c3-0c1s5n0
10.10.100.112    c3-0c1s5n1
10.10.100.113    c3-0c1s5n2
10.10.100.114    c3-0c1s5n3
10.10.100.115    c3-0c2s4n0
10.10.100.116    c3-0c2s4n1
10.10.100.117    c3-0c2s4n2
10.10.100.118    c3-0c2s4n3
```

Example 113. Sample *client-system*.ib file using multiple IB interfaces per router

```
#
# This is the /etc/hosts-like file for Infiniband IP addresses
# on "crystal".
#
10.10.100.101   c0-0c0s1n2
10.10.100.102   c0-0c0s1n2(ib2)
10.10.100.103   c3-0c0s0n1
10.10.100.104   c3-0c0s0n1(ib2)
10.10.100.105   c3-0c0s0n2
10.10.100.106   c3-0c0s0n2(ib2)
10.10.100.107   c2-0c0s1n1
10.10.100.108   c2-0c1s1n1
```

11.9.1.4 *cluster-name*.ib

The *cluster-name*.ib file contains Lustre server InfiniBand IP addresses to cluster (for example, Sonexion) host name mapping information in a /etc/hosts style format. This file must be created by an administrator.

Example 114. Sample *cluster-name*.ib file: *snx11029n*.ib

```
#
# This is the /etc/hosts-like file for Infiniband IP addresses
# on the Sonexion known as "snx11029n".
#
10.10.100.1    snx11029n000    #mgmnt
10.10.100.2    snx11029n001    #mgmnt
10.10.100.3    snx11029n002    #mgs
10.10.100.4    snx11029n003    #mds
10.10.100.5    snx11029n004    #first oss, oss0
10.10.100.6    snx11029n005
10.10.100.7    snx11029n006
10.10.100.8    snx11029n007
10.10.100.9    snx11029n008
10.10.100.10   snx11029n009
10.10.100.11   snx11029n010
10.10.100.12   snx11029n011
10.10.100.13   snx11029n012
10.10.100.14   snx11029n013
10.10.100.15   snx11029n014
10.10.100.16   snx11029n015    #last oss, oss11
```

11.9.1.5 *client-system*.rtrIm

The *client-system*.rtrIm file contains output from the rtr -Im command as executed from the SMW. When capturing the command output to a file, use the -H option to remove the header information from rtr -Im or open the file after capturing and delete the first two lines.

Procedure 93. Creating the *client-system*`.rtrIm` file on the SMW

1. Log on to the SMW.

   ```
   crayadm@boot:~> ssh smw
   Password:
   Last login: Sun Feb 24 23:05:29 2013 from boot
   crayadm@hera-smw:~>
   ```

2. Run the following command to capture the `rtr -Im` output (without header information) to a file:

   ```
   crayadm@hera-smw:~> rtr -Im -H > hera.rtrIm
   ```

3. Move the *hera*`.rtrIm` file to the working directory that you will later run the `clcvt` command from.

   ```
   crayadm@hera-smw:~> mv hera.rtrIm /path/to/working/dir/.
   ```

Example 115. Sample *client-system*`.rtrIm` file: *hera*.rtrIm

```
0        0       c0-0c0s0n0    c0-0c0s0g0     0  0  0
1        1       c0-0c0s0n1    c0-0c0s0g0     0  0  0
2        4       c0-0c0s1n0    c0-0c0s1g0     0  0  1
3        5       c0-0c0s1n1    c0-0c0s1g0     0  0  1
4        8       c0-0c0s2n0    c0-0c0s2g0     0  0  2
5        9       c0-0c0s2n1    c0-0c0s2g0     0  0  2
6        12      c0-0c0s3n0    c0-0c0s3g0     0  0  3
7        13      c0-0c0s3n1    c0-0c0s3g0     0  0  3
8        16      c0-0c0s4n0    c0-0c0s4g0     0  0  4
9        17      c0-0c0s4n1    c0-0c0s4g0     0  0  4
10       20      c0-0c0s5n0    c0-0c0s5g0     0  0  5
11       21      c0-0c0s5n1    c0-0c0s5g0     0  0  5
12       24      c0-0c0s6n0    c0-0c0s6g0     0  0  6
13       25      c0-0c0s6n1    c0-0c0s6g0     0  0  6
14       28      c0-0c0s7n0    c0-0c0s7g0     0  0  7
15       29      c0-0c0s7n1    c0-0c0s7g0     0  0  7
30       32      c0-0c0s0n2    c0-0c0s0g1     0  1  0
31       33      c0-0c0s0n3    c0-0c0s0g1     0  1  0
28       36      c0-0c0s1n2    c0-0c0s1g1     0  1  1
29       37      c0-0c0s1n3    c0-0c0s1g1     0  1  1
...
```

11.9.2 Generating `ip2nets` and `routes` Information

Once you have created and gathered your prerequisite files, you can generate the `persistent-storage` file with the `clcvt generate` action; this portable file will then be used to create `ip2nets` and `routes` directives for the servers, routers, and clients.

The following procedures frequently use the `--split-routes=4` flag, which will print information that can be loaded into `ip2nets` and `routes` files. This method of adding `modprobe.conf` directives is particularly valuable for large systems where the directives might otherwise exceed the `modprobe` buffer limit.

Procedure 94. Creating the `persistent-storage` file

1. Move all of your prerequisite files to an empty directory on the boot node or SMW (the `clcvt` command is only available on the boot node or the SMW). Your working directory should look similar to this when you are ready:

```
crayadm@hera-smw:~/working_dir>  ll
total 240
-rw-rw-r-- 1 crayadm crayadm  23707 Feb  8 14:27 hera.hosts
-rw-rw-r-- 1 crayadm crayadm    548 Feb  8 14:27 hera.ib
-rw-rw-r-- 1 crayadm crayadm  36960 Feb  8 14:27 hera.rtrIm
-rw-rw-r-- 1 crayadm crayadm   1077 Feb  8 14:27 info.snx11029
-rw-rw-r-- 1 crayadm crayadm    662 Feb  8 14:27 snx11029n.ib
```

2. Create the `persistent-storage` file.

```
crayadm@hera-smw:~/working_dir>  clcvt generate
```

Note: The `clcvt` command does not print to `stdout` with successful completion; however, if there are errors when you run the command, you can add debugging information with the `--debug` flag.

Procedure 95. Create `ip2nets` and `routes` information for the compute nodes

1. Execute the `clcvt` command with the `compute` flag to generate directives for the compute nodes.

```
crayadm@hera-smw:~/working_dir>  clcvt compute --split-routes=4
# Place the following line(s) in the appropriate 'modprobe' file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
options lnet ip2nets=/path/to/ip2nets-loading/filename
options lnet routes=/path/to/route-loading/filename
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate ip2nets-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
gni1 10.128.*.*
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate route-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
o2ib6000 1 [68,90]@gni1
o2ib6002 1 [750,751,752,753]@gni1
o2ib6003 1 [618,619,628,629]@gni1
o2ib6004 1 [608,609,638,639]@gni1
o2ib6005 1 [648,649,662,663]@gni1
o2ib6000 2 [608,609,618,619,628,629,638,639,648,649,662,663,750,751,752,753]@gni1
o2ib6002 2 [608,609,638,639]@gni1
o2ib6003 2 [648,649,662,663]@gni1
o2ib6004 2 [750,751,752,753]@gni1
o2ib6005 2 [618,619,628,629]@gni1
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

2. Follow the procedures in Procedure 92 on page 335 to update the compute node boot image `modprobe` information using the `ip2nets` and `routes` information produced by the previous step.

Procedure 96. Create `ip2nets` and `routes` information for service node Lustre clients (MOM and internal login nodes)

1. Execute the `clcvt` command with the `login` flag to generate directives for the service node Lustre clients.

```
crayadm@hera-smw:~/working_dir>  clcvt login --split-routes=4
# Place the following line(s) in the appropriate 'modprobe' file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
options lnet ip2nets=/path/to/ip2nets-loading/filename
options lnet routes=/path/to/route-loading/filename
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate ip2nets-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
gni1 10.128.*.*
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate route-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
o2ib6000 1 [68,90]@gni1
o2ib6002 1 [750,751,752,753]@gni1
o2ib6003 1 [618,619,628,629]@gni1
o2ib6004 1 [608,609,638,639]@gni1
o2ib6005 1 [648,649,662,663]@gni1
o2ib6000 2 [608,609,618,619,628,629,638,639,648,649,662,663,750,751,752,753]@gni1
o2ib6002 2 [608,609,638,639]@gni1
o2ib6003 2 [648,649,662,663]@gni1
o2ib6004 2 [750,751,752,753]@gni1
o2ib6005 2 [618,619,628,629]@gni1
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

2. Follow the procedures in Procedure 89 on page 333 to update the `modprobe` information for the default view of the shared root using the `ip2nets` and `routes` information produced by the previous step.

Procedure 97. Create `ip2nets` and `routes` information for the LNET router nodes

1. Execute the `clcvt` command with the `router` flag to generate directives for the LNET router nodes.

```
crayadm@hera-smw:~/working_dir>  clcvt router --split-routes=4
# Place the following line(s) in the appropriate 'modprobe' file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
options lnet ip2nets=/path/to/ip2nets-loading/filename
options lnet routes=/path/to/route-loading/filename
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate ip2nets-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
gni1 10.128.*.*
o2ib6000 10.10.100.[101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118]
o2ib6002 10.10.100.[103,104,105,106,107,108,109,110]
o2ib6003 10.10.100.[111,112,113,114,115,116,117,118]
o2ib6004 10.10.100.[103,104,105,106,107,108,109,110]
o2ib6005 10.10.100.[111,112,113,114,115,116,117,118]
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate route-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
o2ib6000 1 [68,90]@gni1
o2ib6002 1 [750,751,752,753]@gni1
o2ib6003 1 [618,619,628,629]@gni1
o2ib6004 1 [608,609,638,639]@gni1
o2ib6005 1 [648,649,662,663]@gni1
o2ib6000 2 [608,609,618,619,628,629,638,639,648,649,662,663,750,751,752,753]@gni1
o2ib6002 2 [608,609,638,639]@gni1
o2ib6003 2 [648,649,662,663]@gni1
o2ib6004 2 [750,751,752,753]@gni1
o2ib6005 2 [618,619,628,629]@gni1
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

2. Follow the procedures in Procedure 89 on page 333 to update the `modprobe` information for the LNET router view of the shared root using the `ip2nets` and `routes` information produced by the previous step.

Procedure 98. Create `ip2nets` and `routes` information for the Lustre server nodes

1. Execute the `clcvt` command with the `server` flag to generate directives for the Lustre server nodes.

```
crayadm@hera-smw:~/working_dir>  clcvt server --split-routes=4
# Place the following line(s) in the appropriate 'modprobe' file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
options lnet ip2nets=/path/to/ip2nets-loading/filename
options lnet routes=/path/to/route-loading/filename
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate ip2nets-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
o2ib6 10.10.100.*
o2ib6000 10.10.100.[3,4]
o2ib6002 10.10.100.[5,7,9]
o2ib6003 10.10.100.[6,8,10]
o2ib6004 10.10.100.[11,13,15]
o2ib6005 10.10.100.[12,14,16]
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Place the following line(s) in the appropriate route-loading file.
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
gni1 1 10.10.100.[101,102]@o2ib6000
gni1 1 10.10.100.[103,104,105,106]@o2ib6002
gni1 1 10.10.100.[111,112,113,114]@o2ib6003
gni1 1 10.10.100.[107,108,109,110]@o2ib6004
gni1 1 10.10.100.[115,116,117,118]@o2ib6005
gni1 2 10.10.100.[103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118]@o2ib6000
gni1 2 10.10.100.[107,108,109,110]@o2ib6002
gni1 2 10.10.100.[115,116,117,118]@o2ib6003
gni1 2 10.10.100.[103,104,105,106]@o2ib6004
gni1 2 10.10.100.[111,112,113,114]@o2ib6005
#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

2. Update the `modprobe` information for your Lustre servers using the `ip2nets` and `routes` information produced by the previous step. For more information, refer to your Lustre server documentation.

Resource Utilization Reporting [12]

Resource Utilization Reporting (RUR) is an administrator tool for gathering statistics on how system resources are being used by applications. RUR is a low-noise, scalable infrastructure that collects compute node statistics before an application runs and again after it completes. The extensible RUR infrastructure allows plugins to be easily written to collect data uniquely interesting to each site. Cray supplied plugins collect a variety of data, including process accounting, energy usage, memory usage, and GPU accounting.

12.1 RUR Basics

When RUR is enabled on a Cray system running CLE, resource utilization statistics are gathered from compute nodes running all applications. RUR runs primarily before an application has started and after it ends, ensuring minimal impact on application performance.

Prior to application runtime, the ALPS prologue script calls an RUR prologue script that, based on enabled plugins, initiates preapplication data staging on all compute nodes used by the application. This staging may involve resetting counters to zero or collecting initial values of counters. Following application completion, the ALPS epilogue script calls an RUR epilogue script that gathers these counters, compares them to the initial values, where applicable, stages the data on the compute nodes, and then transfers data from the compute nodes to the login/MOM node. RUR post-processes the data to create a summary report that is written out to a log file or other backing store.

12.1.1 Plugin Architecture

RUR supports a plugin architecture, allowing many types of usage data to be collected while using the same software infrastructure. Two basic types of RUR plugins are supported: *data plugins*, which collect particular statistics about system resources, and *output plugins*, which send the output of the RUR software stack to a backing store.

Cray supplies plugins as part of the RUR distribution, including six data collection plugins, three output plugins, and one example plugin. Sites choose which plugins to enable or disable by modifying the RUR configuration file. See Enable/Disable Plugins on page 351 for more information. Sites can also create custom plugins, specific to their needs, as described in Create Custom RUR Plugins on page 362.

12.1.2 RUR Configuration File

The RUR configuration file `/etc/opt/cray/rur/rur.conf` is located on the shared root. The file consists of distinct sections, identified by a header, [*section*], that contain settings for specific RUR components. Changing the behavior of RUR components is possible through modification of the config file. Configurable values fall into five categories:

- Script/binary location
- Temporary data storage location
- Component timeout specs
- Enabling/disabling plugins
- Optional plugin behavior

Changes to the configuration file are automatically propagated to the nodes via the shared-root mount, requiring no administrator intervention or system reboot. An example configuration file `rur.config.example` is distributed with the RPM.

12.2 RUR Administration

Administration of RUR is primarily accomplished through the RUR configuration file.

12.2.1 Enable RUR

By default, RUR is installed but not enabled during the CLE installation process. Although RUR is not a part of ALPS, it is initiated through the ALPS prologue and epilogue scripts.

Procedure 99. Enable RUR through ALPS

1. Edit the `/etc/opt/cray/alps/alps.conf` file on the shared root such that the `apsys` section includes these variable definitions:

```
apsys
       prologPath      /opt/cray/rur/default/bin/rur_prologue.py
       epilogPath      /opt/cray/rur/default/bin/rur_epilogue.py
       prologTimeout   60
       epilogTimeout   60
/apsys
```

2. Restart ALPS on the login node(s).

```
login:~ # /etc/init.d/alps restart
```

3. Verify that the `rur` argument is set to `True`, `yes`, `1`, or `enable` within the `[global]` section of the RUR configuration file.

```
[global]
rur: True
```

12.2.2 Disable RUR

Disable RUR by setting the `rur` argument to `False`, `no`, `0`, or `disable` within the `[global]` section of the RUR configuration file.

Procedure 100. Disable RUR

1. Edit `/etc/opt/cray/rur/rur.conf` on the shared root.
2. To disable RUR, set `rur` to `False`, `no`, `0`, or `disable` within the `[global]` section.

   ```
   [global]
   rur: false
   ```

12.2.3 Enable/Disable Plugins

Procedure 101. Enable/disable plugins

1. Edit the `/etc/opt/cray/rur/rur.conf` file on the shared root.
2. To enable a plugin, list it in the `[plugins]` section (for data plugins) or `[outputplugins]` section (for output plugins) and set to `true`.

   ```
   [plugins]
   taskstats: true
   ```

3. To disable a plugin, either remove it from the list or set it to `false`.

   ```
   [plugins]
   taskstats: false
   ```

   For example, the following `[plugins]` section from the configuration file explicitly enables data plugins `taskstats` and `timestamp`, but disables `gpustat`. Additionally, plugins `energy`, `kncstats`, and `memory` are implicitly disabled by their absence from the list.

   ```
   [plugins]
   gpustat: false
   taskstats: true
   timestamp: true
   ```

12.2.4 Enable Plugin Options

Many Cray supplied plugins include options that change the plugin's behavior, such as amount of data reported or location of an output file. Follow the procedure to enable a plugin's optional behavior.

Procedure 102. Enable plugin options

1. Edit the `/etc/opt/cray/rur/rur.conf` file on the shared root.

2. Insert the line:

   `arg:` *option*

   in the [*plugin*] section, where *option* is the plugin-specific value for the desired behavior and *plugin* is the appropriate plugin name.

   For example, to select `memory` plugin's `extended_buddy` option, add `arg: extended_buddy` in the `[memory]` section as shown.

```
[memory]
stage:  /opt/cray/rur/default/bin/memory_stage.py
post:   /opt/cray/rur/default/bin/memory_post.py
arg:  extended_buddy
```

3. Verify that the desired plugin has been enabled in either the `[plugins]` or `[outputplugins]` section of the RUR configuration file, as described in Enable/Disable Plugins on page 351.

12.2.5 RUR Debugging

The value of `debug_level`, if set, within the `[global]` section of the configuration file determines the level of debug verbosity written to the messages file on the SMW. Valid values are `debug`, `info`, `warning error`, and `critical`.

12.3 Included Data Plugins

Cray supplied data plugins collect a variety of data. Sites may develop other data plugins to collect information not supported by Cray plugins, as described in Create Custom RUR Plugins on page 362.

12.3.1 `energy`

The `energy` plugin collects compute node energy usage data. The amount of data reported and the format in which it is written is determined by the value of the argument `arg` set in the `[energy]` section of the RUR configuration file.

If `arg` is not set (default) or set to `json-list`, the plugin reports the following, written in JavaScript Object Notation (JSON) list format:

`energy_used`

> The total energy (joules) used across all nodes. On nodes with accelerators, this value includes `accel_energy_used`, the total energy used by the accelerators.

The following example shows default `energy` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

Example 116. RUR default energy ouput

```
2013-08-30T11:19:06.545114-05:00 c0-0c0s2n2 RUR 18657 p2-20130829t090349 [RUR@34]
uid: 12345, apid: 10963, jobid: 0, cmdname: /opt/intel/vtune_xe_2013/bin64/amplxe-cl
plugin: energy ['energy_used', 318]
```

If `arg` is set to `json-dict`, the plugin also reports the following extended energy data, written in JSON dictionary format:

`nodes` Number of nodes in job

`nodes_power_capped`

Number of nodes with nonzero power cap

`nodes_throttled`

Number of nodes that experienced one or more throttled sockets

`nodes_with_changed_power_cap`

Number of nodes with power caps that changed during execution. On nodes with accelerators, this value includes the number of accelerators with power caps that changed.

`max_power_cap`

Maximum nonzero power cap

`max_power_cap_count`

Number of nodes with the maximum nonzero power cap

`min_power_cap`

Minimum nonzero power cap

`min_power_cap_count`

Number of nodes with the minimum nonzero power cap

On nodes with accelerators, the extended data also include the following data:

`accel_energy_used`

Total accelerator energy (joules) used

`nodes_accel_power_capped`

Number of accelerators with nonzero power cap

`max_accel_power_cap`

Maximum nonzero accelerator power cap

`max_accel_power_cap_count`

Number of accelerators with the maximum nonzero power cap

`min_accel_power_cap`

Minimum nonzero accelerator power cap

`min_accel_power_cap_count`

Number of accelerators with the minimum nonzero power cap

The following example shows extended `energy` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

Example 117. RUR extended `energy` output

```
2014-01-17T10:05:54.026557-06:00 c0-0c0s1n1 RUR 11674 p0-20140116t214834 [rur@34]
uid: 12345, apid: 286342, jobid: 0, cmdname: /bin/cat, plugin: energy {"energy_used": 5641,
"accel_energy_used": 1340, "nodes": 32, "nodes_power_capped": 3, "min_power_cap": 155,
"min_power_cap_count": 2, "max_power_cap": 355, "max_power_cap_count": 1,
"nodes_accel_power_capped": 3, "min_accel_power_cap: 200, min_accel_power_cap_count": 3,
"max_accel_power_cap": 200, "max_accel_power_cap_count": 3, "nodes_throttled": 0,
"nodes_with_changed_power_cap": 0}
```

12.3.2 `gpustat`

The `gpustat` plugin collects the following utilization statistics for NVIDIA GPUs, if present. The data is written in JSON list format.

`maxmem` Maximum memory used across all nodes

`summem` Total memory used across all nodes

`gpusecs` `gpusecs`

The following example shows `gpustat` data as written in `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

Example 118. RUR gpustat output

```
2013-07-09T15:50:42.761257-05:00 c0-0c0s2n2 RUR 11329 p2-20130709t145714 [RUR@34]
uid: 12345, apid: 8410, jobid: 0, cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000,
'summem', 108000, 'gpusecs', 44]
```

12.3.3 kncstats

The `kncstats` plugin collects the following process accounting data from Intel Xeon Phi (KNC) coprocessors, if present. The data is written in JSON list format.

`core`	Set to `1` if core dump occurred
`exitcode`	Lists all unique exit codes
`max_rss`	Maximum memory used
`rchar`	Characters read by process
`stime`	System time
`utime`	User time
`wchar`	Characters written by process

The following example show `kncstats` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

Example 119. RUR kncstats output

```
2014-02-25T18:49:12.101383-06:00 c0-0c0s1n1 RUR 11274 p0-20140225t135439 [RUR@34]
uid: 12345, apid: 539224, jobid: 0, cmdname: /bin/date, plugin: kncstats ['utime', 8000,
'stime', 20000, 'max_rss', 620, 'rchar', 2730, 'wchar', 119, 'exitcode:signal',
['0:0'], 'core', 0]
```

12.3.4 memory

The `memory` plugin collects information from `/proc` and `/sys` that is useful when assessing an application's memory performance. The data is written in JSON dictionary format. The amount of data reported is determined by the value of the argument `arg` set in the `[memory]` section of the RUR configuration file.

If `arg` is not set (default), the plugin reports the following data:

`%_of_boot_mem`

The % of boot memory for each order chunk in `/proc/buddyinfo` summed across all memory zones

`Active(anon)`

Total amount of memory in active use by the application

`Active(file)`

Total amount of memory in active use by cache and buffers

`boot_freemem`

Contents of `/proc/boot_freemem`

`current_freemem`

Contents of `/proc/current_freemem`

`free` Number of hugepages that are not yet allocated

`hugepages-`*size*`kB`

The hugepage size for the select entries from `/sys/kernel/mm/hugepages/hugepages-*kB/*`

`Inactive(anon)`

Total amount of memory that is candidate to be swapped out

`Inactive(file)`

Total amount of memory that is candidate to be dropped from cache

`nr` Number of hugepages that exist at this point

`resv` Number of hugepages committed for allocation, but no allocation has occurred

`Slab` Total amount of memory used by the kernel

`surplus` Number of hugepages above `nr`

Example 120. RUR default `memory` output

```
2014-03-21T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710 p0-20140321t091957 [RUR@34]
uid: 12345, apid: 33079, jobid: 0, cmdname: /bin/hostname, plugin: memory
{"current_freemem": 21858372, "meminfo": {"Active(anon)": 35952, "Slab": 105824,
"Inactive(anon)": 1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"%_of_boot_mem": ["67.23", "67.23", "67.23", "67.22", "67.21", "67.18", "67.11"
, "67.04", "66.94", "66.83", "66.77", "66.66", "66.53", "66.38", "65.87", "65.07
", "63.05", "61.43"], "nid": "8", "cname": "c0-0c0s2n0", "boot_freemem": 3243262
8}
```

If `arg` is set to `extended_buddy`, the output relating to `/proc/buddyinfo` includes NUMA node granularity information in addition to the existing node granularity information. This information is useful when troubleshooting certain fragmentation related issues.

The following examples show `memory` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

Example 121. RUR extended `memory` output

```
2014-03-21T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710 p0-20140321t091957 [RUR@34]
uid: 12345, apid: 33079, jobid: 0, cmdname: /bin/hostname, plugin: memory
{"current_freemem": 21858372, "meminfo": {"Active(anon)": 35952, "Slab": 105824,
"Inactive(anon)": 1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"Node_0_zone_DMA": ["0.05", "0.05", "0.05", "0.05", "0.05", "0.05", "0.05", "0.05",
"0.05", "0.04", "0.04", "0.03", "0.00", "0.00", "0.00", "0.00", "0.00", "0.00"],
"%_of_boot_mem": ["67.23", "67.23", "67.23", "67.22", "67.21", "67.18", "67.11"
, "67.04", "66.94", "66.83", "66.77", "66.66", "66.53", "66.38", "65.87", "65.07
", "63.05", "61.43"], "nid": "8", "cname": "c0-0c0s2n0", "boot_freemem": 3243262
8, "Node_0_zone_DMA32": ["6.07", "6.07", "6.07", "6.07", "6.07", "6.07", "6.07",
 "6.06", "6.05", "6.04", "6.01", "5.94", "5.86", "5.76", "5.46", "4.85", "3.23",
 "3.23"], "Node_0_zone_Normal": ["61.11", "61.11", "61.11", "61.11", "61.09", "6
1.07", "60.99", "60.93", "60.84", "60.75", "60.72", "60.70", "60.67", "60.62", "
60.42", "60.22", "59.81", "58.20"]}
```

12.3.5 `taskstats`

The `taskstats` plugin collects process accounting data. The amount of data reported and the format in which it is written is determined by the value of the argument `arg` set in the `[taskstats]` section of the RUR configuration file.

If `arg` is not set (default), the plugin reports the following basic process accounting data similar to that provided by UNIX process accounting or `getrusage`. These values are sums across all nodes, except for the memory used, which is the maximum value across all nodes. The data is written in JSON list format.

`core`	Set to `1` if core dump occurred
`exitcode`	Lists all unique exit codes
`max_rss`	Maximum memory used
`rchar`	Characters read by process
`stime`	System time
`utime`	User time
`wchar`	Characters written by process

Example 122. RUR default `taskstats` output

For a job that exits normally:

```
2013-11-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417 p0-20131101t153028 [RUR@34]
uid: 12345, apid: 86989, jobid: 0, cmdname: /lus/tmp/rur01.2338/./CPU01-2338
plugin: taskstats ['utime', 10000000, 'stime', 0, 'max_rss', 940, 'rchar', 107480,
'wchar', 90, 'exitcode:signal', ['0:0'], 'core', 0]
```

For a job that core dumps:

```
2013-11-02T11:12:45.020716-05:00 c0-0c1s1n2 RUR 3731 p0-20131101t153028 [RUR@34]
uid: 12345, apid: 86996, jobid: 0, cmdname: /lus/tmp/rur01.3657/./exit04-3657
plugin: taskstats ['utime', 4000, 'stime', 144000, 'max_rss', 7336, 'rchar', 252289,
'wchar', 741, 'exitcode:signal', ['0:9', '139:0', '0:11', '0:0'], 'core', 1]
```

If `arg` is set to `xpacct`, the plugin also provides the following extended process accounting data similar to that which was collected by the deprecated Cray System Accounting (CSA).

`abortinfo`	If abnormal termination occurs, a list of `abort_info` fields is reported
`apid`	Application ID as defined by application launcher
`bkiowait`	Total delay time (ns) waiting for synchronous block I/O to complete
`btime`	UNIX time when process started
`comm`	String containing process name. May be different than the header, which is the process run by the launcher.
`corememm`	Integral of RSS used by process in MB-usecs[1]
`ecode`	Process exit code
`etime`	Total elapsed time in microseconds
`gid`	Group ID
`jid`	Job ID — the PAGG job container used on the compute node
`majfault`	Number of major page faults
`minfault`	Number of minor page faults
`nice`	POSIX `nice value` of process
`nid`	String containing node ID
`pgswapcnt`	Number of pages swapped; should be `0` on Cray compute nodes
`pid`	Process ID

[1] The current memory usage is added to these counters (i.e., `coremem`, `vm`) every time. A tick is charged to a task's system time. Therefore, at the end we will have memory usage multiplied by system time and an average usage per system time unit can be calculated.

`pjid`	Parent job ID — the PAGG job container on the MOM node
`ppid`	Parent process ID
`prid`	Job project ID
`rcalls`	Number of read system calls
`rchar`	Characters read by process
`rss`	RSS highwater mark
`sched`	Scheduling discipline used on node
`uid`	User ID
`vm`	Integral of virtual memory used by process in MB-usecs[1]
`wcalls`	Number of write system calls
`wchar`	Characters written by process

Example 123. RUR extended `taskstats` output

```
2013-10-18T10:29:38.285378-05:00 c0-0c0s1n1 RUR 24393 p1-20131018t081133 [RUR@34]
uid: 12345, apid: 370583, jobid: 0, cmdname: /bin/cat, plugin: taskstats
['btime', 1386061749, 'etime', 8000, 'utime', 0, 'stime', 4000, 'coremem', 442,
'max_rss', 564, 'max_vm', 564, 'pgswapcnt', 63, 'minfault', 15, 'majfault', 48,
'rchar', 2608, 'wchar', 686, 'rcalls', 19, 'wcalls', 7, 'bkiowait', 1000,
'exitcode:signal', [0], 'core', 0]
```

If `arg` is set to `xpacct, per-process`, the plugin reports extended accounting data for every compute node process rather than a summary of all processes for an application. `per-process` must be set in combination with `xpacct`.

Caution: If `per-process` is set and many processes are run on each node, the volume of data generated and stored on disk can become an issue.

Example 124. RUR per-process `taskstats` output

This output was generated with the `json-dict` option set.

```
2013-12-03T13:25:34.446167-06:00 c0-0c2s0n2 RUR 7623 p3-20131202t090205 [RUR@34]
uid: 12345, apid: 1560, jobid: 0, cmdname: ./it.sh, plugin: taskstats {"uid": 12795,
"wcalls": 37, "pid": 2997, "vm": 16348, "jid": 395136991233, "bkiowait": 1201616,
"majfault": 1, "etime": 0, "btime": 1386098731, "gid": 0, "ppid": 2992, "utime": 0,
"nice": 0, "sched": 0, "nid": "92", "prid": 0, "comm": "mount", "stime": 4000,
"wchar": 3465, "rss": 1028, "minfault": 352, "coremem": 1109, "ecode": 0,
"rcalls": 22, "pjid": 7045, "pgswapcnt": 0, "rchar": 12208}

2013-12-03T13:25:34.949138-06:00 c0-0c2s0n2 RUR 7623 p3-20131202t090205 [RUR@34]
uid: 12345, apid: 1560, jobid: 0, cmdname: ./it.sh, plugin: taskstats {"uid": 12795,
"wcalls": 0, "pid": 2998, "vm": 20268, "jid": 395136991233, "bkiowait": 0,
"majfault": 0, "etime": 0, "btime": 1386098731, "gid": 0, "ppid": 2992, "utime": 0,
"nice": 0, "sched": 0, "nid": "92", "prid": 0, "apid": 1560, "comm": "ls", "stime":
4000, "wchar": 0, "rss": 1040, "minfault": 360, "coremem": 3140, "ecode":
0, "rcalls": 19, "pjid": 7045, "pgswapcnt": 0, "rchar": 10629}
```

If `arg` is set to `json-dict`, the data is written in JSON dictionary format.

If `arg` is set to `json-list`, the data is written in JSON list format (default).

12.3.6 `timestamp`

The `timestamp` plugin collects the start and end times of an application. The following example shows `timestamp` data, as written in `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW, for an application that slept 20 seconds.

Example 125. RUR `timestamp` data

```
2013-08-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882 p3-20130830t074847 [RUR@34] uid: 12345,
apid: 6640, jobid: 0, cmdname: /bin/sleep plugin: timestamp APP_START
2013-08-30T14:31:46CDT APP_STOP 2013-08-30T14:32:06CDT
```

12.4 Included Output Plugins

Cray supplied output plugins support Lightweight Log Manager (LLM) files and flat text files as the backing store for RUR data. Sites may develop other output plugins to satisfy site-specific output needs not supported by Cray plugins, as described in Create Custom RUR Plugins on page 362.

12.4.1 `file`

The `file` plugin allows RUR data to be stored to a flat text file on any file system to which the login node can write. This plugin is also intended as a very simple guide for anyone interested in writing an output plugin.

The following is sample output from `file` to a location defined in the RUR configuration file:

```
uid: 1000, apid: 8410, jobid: 0, cmdname: /tmp/dostuff plugin: taskstats ['utime', 32000,
'stime', 132000, 'max_rss', 1736, 'rchar', 44524, 'wchar', 289] uid: 1000, apid: 8410,
jobid: 0, cmdname: /tmp/dostuff plugin: energy ['energy_used', 24551] uid: 1000,
apid: 8410, jobid: 0, cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000,
'summem', 108000]
```

12.4.2 `llm`

The `llm` plugin aggregates log messages from various Cray nodes and places them on the SMW. `llm` has its own configuration options, but typically it will place RUR messages into the messages log file `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW. The messages shown in the previous sections are in LLM log format.

12.4.3 user

The `user` plugin writes RUR output for a user's application to the user's home directory (default) or a user-defined location, only if the user has indicated that this behavior is desired. See User Options on page 361.

The naming of the default output file(s), `rur.`*suffix*, is dependent on the value of the argument `arg`, which defines a report type and is set in the `user` section of the RUR configuration file. If `arg` is set to:

`apid` An output file is created for each application executed and *suffix* is the `apid`.

`jobid` An output file is created for each job submitted and *suffix* is the `jobid`.

`single` All output is placed in a single file and no suffix is appended to the output file name.

12.4.3.1 User Options

Users have the option to opt-in or out for the user plugin, redirect plugin output to a specific file or directory, or override the default report type.

- By default, RUR data is not written to a user's directory. A user must either create the file `~/.rur/user_output_optin` to indicate that data should be written, or create a file that initiates one of the following two options.
- Users may redirect the output of RUR by specifying a redirect location in `~/.rur/user_output_redirect`. The contents of this file must be a single line that specifies the absolute or relative (from the user's home directory) path of the directory or file to which the RUR output data is to be written. If the redirect file either does not exist, points to a path that does not exist, or points to a path to which the user does not have write permission, then the output is written to the user's home directory.

  A user with an existing `~/.rur/user_output_redirect` file can temporarily stop RUR data from being written by setting the redirect path to `/dev/null`.
- Additionally, the user may override the default report type by specifying a valid report type in `~/.rur/user_output_report_type`. Valid report types are `apid`, `jobid`, or `single`, resulting in the user's RUR data being written to one file per application, one file per job, or a single file, respectively. If the file `~/.rur/user_output_report_type` is empty or contains an invalid type, then the default report type, as defined in the configuration file, is created.

12.5 Included Example Plugins

Plugins included in this section are provided as guides for sites wanting to develop similar plugins geared towards site-specific needs.

12.5.1 database

The `database` plugin is provided as a guide for sites wanting to output RUR data to a site-supplied database. Sites will need to configure their own systems, provide an external database, create their own tables, and modify `database_output.py` to collect the desired data.

MySQL is the database supported by the example plugin. The following arguments are defined for connecting to a database:

- `DB_NAME='rur'`
- `DB_USER='rur_user'`
- `DB_PASS='rur_pass'`
- `DB_HOST='rur_host'`

The `database` plugin collects the values: `energy_used`, `apid`, `jobid`, and `uid`, and saves this data to a table, `energy`. It does this by performing the following:

- Digests RUR data into a dictionary and saves it to class `DbData`
- Creates rules for saving data collected in `DbData` to particular tables
- Uses the rules to scan the `DbData` dictionary and `INSERT` that data into a database

Cray recommends that the database is not hosted on SDB or login nodes. It should also be noted that, depending on job load, interacting with an external database may cause system latency.

12.6 Create Custom RUR Plugins

RUR's plugin architecture enables sites to create custom plugins for their specific interests and needs. The following sections describe the components and requirements of valid plugins and how to implement new plugins within the RUR infrastructure.

12.6.1 Data Plugins

A data plugin is comprised of a *staging component* and a *post processing component*. The data plugin staging component is called by `rur-stage.py` on the compute node prior to the application running and again after the application has completed. The staging component may reset counters before application execution and collect them after application completion, or it may collect initial and final values prior to and after application execution, respectively, and then calculate the delta values. Python functions have been defined to simplify writing plugins, although it is not necessary for the plugin to be written in Python. The interface for the data plugin staging component is through command line arguments.

12.6.1.1 Data Plugin Staging Component

All data plugin staging components must support the following arguments:

`--apid`=*apid*

Defines the application ID of the running application.

`--timeout`=*time*

Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.

`--pre` Indicates the plugin is being called prior to the application.

`--post` Indicates the plugin is being called after the application.

`--outputfile`=*output_file*

Defines where the output data is written. Each plugin should define a default output file in `/var/spool/RUR/` if this argument is not provided.

`--arg`=*arg* A plugin-specific argument, set in the RUR config file. RUR treats this as an opaque string.

The output of an RUR data plugin staging component is a temporary file located in `/var/spool/RUR` on the compute node. The file name must include both the name of the plugin, as defined in the RUR config file, and *.apid*. The RUR gather phase will automatically gather the staged files from all compute nodes after the application has completed and place it in `gather_dir` as defined in the configuration file.

The following is an example of a simple data plugin staging component:

Example 126. Data plugin staging component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin staging component
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
   apid, inputfile, outputfile, timeout, pre, post, \
      parg = rur_plugin_args(sys.argv[1:])
   if outputfile is "":
       outputfile = "/var/spool/RUR/pluginname."+str(apid)
   if (pre==1):
       zero_counters()
   else:
       write_postapp_stateto(outputfile)

if __name__ == "__main__":
   main()
```

12.6.1.2 Data Plugin Post Processing Component

A data plugin also requires a post processing component that processes the data staged by the staging component and collected during the RUR gather phase. The post processing component is called by `rur-post.py`. The input file contains records, one node per line, of all of the statistics created by the staging component. The output of the post processing component is a file containing the summary of data from all compute nodes.

All data plugin post processing components must support the following arguments:

`--apid`=*apid*

Defines the application ID of the running application.

`--timeout`=*time*

Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.

`--inputfile`=*input_file*

Specifies the file from which the plugin gets its input data.

`--outputfile`=*output_file*

Specifies the file to which the plugin writes its output data.

The following is an example of a simple data plugin post processing component:

Example 127. Data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin post processing component
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile is "":
       outputfile = inputfile + ".out"

   pc = PostCompute()
   pc.process_file(inputfile)
   formated = pc.present_entries([('plugin_foo_data','sum')])
   fout=open(outputfile, 'w+')
   fout.write("energy %s" % formated)

if __name__ == "__main__":
   main()
```

12.6.2 Output Plugins

Output plugins allow RUR data to be outputted to an arbitrary backing store. This can be a storage device or another piece of software that then consumes the RUR data. The output plugin is passed a number of command line arguments that describe the application run and provide a list of input working files (the output of data plugin post processing components). The plugin takes the data in the working files and exports it to the destination specified in the RUR configuration file for the specific output plugin.

Tip: If there is an error from an output plugin, the error message appears in the ALPS log `/var/opt/cray/alps/log/apsys` on the service node rather than the LLM logs on the SMW.

The following is an example of a simple output plugin.

Example 128. Output plugin

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample output plugin
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_output_args

def main():
   apid, jobid, uid, cmdname, inputfilelist, timeout, \
   parg = rur_output_args(sys.argv[1:])

   outfile = open(parg, "a")
   for inputfile in inputfilelist:
       infile =  open(inputfile, "r")
       lines = infile.readlines()
       for line in lines:
           outfile.write(line)
       infile.close()
   outfile.close()
```

12.6.3 Implement a New RUR Plugin

For a site written plugin to run, it must be added to the RUR configuration file and enabled. Follow the procedure to define and configure a new plugin.

Procedure 103. Modify RUR to define and configure a site written plugin

1. Ensure that the site written plugin is located on a file system that is readable by compute nodes, owned by `root`, and not writeable by non-`root` users.
2. Add a new plugin definition section to the RUR configuration file:
   a. If adding a data plugin, the definition section must include: the plugin name, a `stage` definition (the complete path to the plugin's data staging script), and a `post` definition (the complete path to the plugin's post processing script).

      For example, to define the site written data plugin `siteplug`, the entry within the RUR configuration would appear as follows:

      ```
      # The siteplug Data Plugin collects data that is
      #     of particular interest to this site.
      # Stage - The staging component run by rur_stage on the
      #     compute node
      # Post - The post-processing component run by rur_post on
      #     the login/mom node
      [SitePlug]
      stage: /opt/cray/rur/default/bin/siteplug_stage.py
      post: /opt/cray/rur/default/bin/siteplug_post.py
      ```

b. If adding an output plugin, the definition section must include: the plugin name, an `output` definition (the complete path to the output plugin script or binary), and an optional argument.

   For example, to define the site written output plugin `siteout`, the entry within the RUR configuration would appear as follows:

```
# The siteout output plugin.
# Write RUR output to a text file on the site's huge
# archive file system.
[siteout]
output:/opt/cray/rur/site/bin/site_output.py
arg:hsmuser@hsmbackup.site.com:/hsmuser/rurbackup
```

3. Add the new plugin to either the data plugin or output plugin configuration section, labeled `[plugins]` or `[outputplugins]`, respectively. Indicate `true` to enable or `false` to disable plugin execution.

```
# Data Plugins section Configuration
# Define the supported Data plugins and enable/disable
# them. Plugins defined as "Plugin: False" will not run,
# but will be parsed for correct config file syntax.
[plugins]
gpustat: true
taskstats: true
SitePlug: true

# Output Plugins section Configuration
# Define which output plugins are supported, and enable/
# disable them. Plugins defined as "Plugin: False" will
# not run, but will be parsed for correct config file
# syntax.
[outputplugins]
llm: true
file: false
siteout: false
```

12.6.4 Additional Plugin Examples

This is a set of RUR plugins that report information about the number of available huge pages on each node. The huge page counts are reported in `/proc/buddyinfo`. There are two versions of the staging component: one that reports what is available and the second that reports changes during the application run.

Example 129. Huge pages data plugin staging component (version A)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
# Each node reports its nid and the number of available pages of each size.
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
    apid, inputfile, outputfile, timeout, pre, post, parg =rur_plugin_args(sys.argv[1:])
    if outputfile == 0:
      outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
    if (pre==1):
      zero_counters()
    else:
      nidf = open("/proc/cray_xt/nid", "r")
      n = nidf.readlines()
      nid = int(n[0])
      inf = open("/proc/buddyinfo", "r")
      b = inf.readlines()
      sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0), ('16M', 0), ('32M', 0), ('64M', 0)])

      for line in b:
        l = line.split()
        sizes['2M'] += int(l[13])
        sizes['4M'] += int(l[14])
        sizes['8M'] += int(l[15])
        sizes['16M'] += int(l[16])
        sizes['32M'] += int(l[17])
        sizes['64M'] += int(l[18])

        o = open(outputfile, "w")
        o.write("{6} {0} {1} {2} {3} {4} {5}".format(sizes['2M'],sizes['4M'], \
        sizes['8M'], sizes['16M'], sizes['32M'], sizes['64M'], nid))
        o.close()

if __name__ == "__main__":
    main()
```

Example 130. Huge pages data plugin staging component (version B)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
# This plugin records the number of available pages before the job is launched.
# At job completion time it reports the change
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
```

```
   apid, inputfile, outputfile, timeout, pre, post, parg =rur_plugin_args(sys.argv[1:])
   if outputfile == 0:
  outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
   if (pre==1):
  inf = open("/proc/buddyinfo", "r")
  b = inf.readlines()
  sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0), ('16M', 0), ('32M', 0), ('64M', 0)])
  for line in b:
    l = line.split()
    sizes['2M'] += int(l[13])
    sizes['4M'] += int(l[14])
    sizes['8M'] += int(l[15])
    sizes['16M'] += int(l[16])
    sizes['32M'] += int(l[17])
    sizes['64M'] += int(l[18])

       o = open("/tmp/buddyinfo_save", "w")
       o.write("{0} {1} {2} {3} {4} {5}".format(sizes['2M'],sizes['4M'], \
   sizes['8M'], sizes['16M'], sizes['32M'], sizes['64M']))
       o.close()
   else:
  nidf = open("/proc/cray_xt/nid", "r")
  n = nidf.readlines()
  nid = int(n[0])
  inf = open("/proc/buddyinfo", "r")
  b = inf.readlines()
  sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0), ('16M', 0), ('32M', 0), ('64M', 0)])

  for line in b:
    l = line.split()
    sizes['2M'] += int(l[13])
    sizes['4M'] += int(l[14])
    sizes['8M'] += int(l[15])
    sizes['16M'] += int(l[16])
    sizes['32M'] += int(l[17])
    sizes['64M'] += int(l[18])

  obf = open("/tmp/buddyinfo_save", "r")
  ob = obf.readlines()
  n=0

  obd0 = ob[0]
  obd = obd0.split()

  diff = [
    (int(obd[0]) - sizes['2M']),
    (int(obd[1]) - sizes['4M']),
    (int(obd[2]) - sizes['8M']),
    (int(obd[3]) - sizes['16M']),
    (int(obd[4]) - sizes['32M']),
    (int(obd[5]) - sizes['64M'])
    ]

       o = open(outputfile, "w")
# uncomment the following line to get the actual sizes
       #o.write("sizes {6} {0} {1} {2} {3} {4} {5}\n".format(sizes['2M'],sizes['4M'], \
   sizes['8M'], sizes['16M'], sizes['32M'], sizes['64M'], nid))
       o.write("diff {6} {0} {1} {2} {3} {4} {5}".format(diff[0], diff[1], diff[2], \
```

```
    diff[3], diff[4], diff[5], nid))
        o.close()
 os.unlink("/tmp/buddyinfo_save")

if __name__ == "__main__":
    main()
```

Example 131. Huge pages data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is a RUR postprocessing pluging for the buddyinfo data
# collection. It copies the input files to output, adding a
# "buddyinfo" label.
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile == 0:
       outputfile = inputfile + ".out"

   fin=open(inputfile, "r")
   l = fin.readlines()

   fout=open(outputfile, 'w+')
   for line in l:
   fout.write("buddyinfo {0}".format(line))

if __name__ == "__main__":
   main()
```

12.7 Migration Tips

This section includes tips on replacing the functionality previously provided by deprecated accounting software.

12.7.1 Application Completion Reporting (ACR)

Cray supplied RUR data plugins collect the same data found in Mazama's Application Completion Reporting (ACR) feature, but RUR does not include a reporting utility like `mzreport`. When using RUR's `llm` output plugin, the type of data reported by `mzreport` can be extracted from the output files as demonstrated in the following sections.

12.7.1.1 ACR Job Reporting

The information provided by `mzreport -j` and `mzreport --job` can easily be obtained in the RUR environment from the log files `/var/opt/cray/log/`*partition*`-current/messages-`*date* by invoking the following command:

```
smw:~ # grep -e "RUR" messages-* |grep -e "jobid: jobid"
```

12.7.1.2 ACR Timespan Reporting

In ACR, `mzreport -t` and `mzreport -T` control the span of time over which job completions are reported. The following example is a simple Python script, `timesearch.py`, that provides this functionality.

```
#cat timesearch.py
#!/usr/bin/env python
for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if (rurline.split(' ')[1] > sys.argv[2]) and (rurline.split(' ')[1] < sys.argv[3]):
       print rurline
```

The script is called with the log file of interest and the desired start/stop time stamps, as follows:

```
smw:~ # python ./timesearch.py messages-date "start_time" "end_time"
```

Where *start_time* and *end_time* are formatted as "*yyyy-mm-dd*T*hh*:*mm*:*ss*".

12.7.1.3 ACR Exit Code Reporting

The `get_exit.py` Python script listed here provides a list of the user IDs with the most non-zero exit codes.

```
# cat get_exit.py
#!/usr/bin/env python
import os,sys,re

statre = re.compile("'(\w*):(\w*)',\s*\[('(\w*):(\w*)'(, )?)+\]")
statsre = re.compile("(\w*):(\w*)")
uidre  = re.compile("uid:\s*(\w*)")
cnt = {}

for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if 'taskstats' in rurline:
       sus = statre.search(rurline)
       status = sus.group()
       stats = statsre.findall(status)
       for stat in stats[1:]:
           if stat[0] != '0':
               uid = int(uidre.findall(rurline)[0])
               if cnt.get(str(uid)):
                   cnt[str(uid)] += 1
               else:
                   cnt[str(uid)] = 1

x = sorted(cnt, key = cnt.get, reverse=True)
print "uids with the most non-zero exit codes %s" % x[:sys.argv[2]]
```

The script is called with the log file of interest and the number of user IDs on which to report, as follows:

```
smw:~ #python ./get_exit.py messages-date num
```

12.7.2 Application Resource Utilization (ARU)

Sites that use ARU will have an easy transition to RUR as all of the data provided in ARU is available in RUR, but in a slightly different format. The ARU output below is available by enabling the `taskstats` plugin's default behavior.

```
2012-11-26T08:52:37.802113-06:00 c0-0c0s0n2 apsys 19864 p0-20121126t060549 -
apid=6240364, Finishing, user=8855, batch_id=114.sdb, exit_code=0, exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3168 inblocks=0 outblocks=0 cpus=8
start=Mon Nov 26 08:52:37 2012 stop=Wed Dec 31 18:00:00 1969 cmd=growfiles
```

RUR `taskstats` default output:

```
2013-11-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417 p0-20131101t153028 [RUR@34]
uid: 10973, apid: 86989, jobid: 0, cmdname: /lus/esfs/overby/rur01.2338/./CPU01-2338
plugin: taskstats ['utime', 10000000, 'stime', 0, 'max_rss', 940, 'rchar', 107480,
'wchar', 90, 'exitcode:signal', ['0:0'], 'core', 0]
```

And the ARU output below is available by enabling the `timestamp` plugin.

```
2012-11-26T08:53:15.618239-06:00 c0-0c0s0n2 apsys 20604 p0-20121126t060549 -
apid=6240378, Finishing,user=8855, batch_id=121.sdb, exit_code=0, exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3152 inblocks=0 outblocks=0 cpus=1
start=Mon Nov 26 08:52:51 2012 stop=Wed Dec 31 18:00:00 1969 cmd=close2_01
```

RUR `timestamp` output:

```
2013-08-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882 p3-20130830t074847 [RUR@34] uid: 0,
apid: 6640, jobid: 0, cmdname: /bin/sleep plugin: timestamp APP_START
2013-08-30T14:31:46CDT APP_STOP 2013-08-30T14:32:06CDT
```

12.7.3 Cray System Accounting (CSA)

The Cray supplied data plugin `taskstats`, when enabled and configured for extended accounting data, collects all of the data in the CSA process accounting record with the exception of `ac_sbu`, the system billing units. Example 123 shows this data as written in `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

RUR does not include the report generation capabilities provided by CSA, however, the type of data reported by CSA can be extracted from the messages files on the SMW. The following is a short Python script for searching through these files. It allows filtering for group ID (`-g`), job ID (`-j`), user ID (`-u`), and system time exceeding a certain value (`-s`); similar to the `csacom` filters `-g`, `-j`, `-u`, `-o`, respectively.

```
#!/usr/bin/env python
# Usage: filter-messages [-g gid] [-j jid] [-u uid] [-s stime] -f messages-date
import os,sys,re,getopt,collections

def getcmdlineargs(args):
   arglist = collections.defaultdict(lambda: 0, {})
   options, remainder = getopt.getopt(args,
       'g:j:u:s:f:',
       ['gid=', 'jid=', 'uid=', 'Stimeexceeds=', 'filename='])

   for opt,arg in options:
       if opt in ('-g', '--gid'):
           arglist['gid'] = arg
       if opt in ('-j', '--jid'):
           arglist['jid'] = arg
       if opt in ('-u', '--uid'):
           arglist['uid'] = arg
       if opt in ('-s', '--Stimeexceeds'):
           arglist['stimeexceeds'] = arg
       if opt in ('-f', '--filename'):
           arglist['filename'] = arg
   return arglist

def reeqgt(tag, restr, rurline, eq):
   retre = re.compile("'" + str(restr) + "'," + "\s*(\w*)")
   field = retre.findall(rurline)
   if field == []:
```

```
        return False
    if eq and tag == field[0]:
        return True
    elif (not eq) and tag <= field[0]:
        return True
    return False

arglist = getcmdlineargs(sys.argv[1:])
if not arglist['filename']:
    exit(1)
for rurline in [line for line in open(arglist['filename'], 'r') if 'RUR' in line]:
  if 'taskstats' in rurline:
        if arglist['jid'] and not (reeqgt(arglist['jid'], 'jid', rurline, 1)):
                continue
        if arglist['uid'] and not (reeqgt(arglist['uid'], 'uid', rurline, 1)):
                continue
        if arglist['gid'] and not (reeqgt(arglist['gid'], 'gid', rurline, 1)):
                continue
        if arglist['stimeexceeds'] and not (reeqgt(arglist['stimeexceeds'], 'stime',
           rurline, 0)):
                continue

        print "%s" % rurline,
```

Configuring Service Node MAMU [13]

This chapter describes how to configure repurposed compute nodes as Multiple Application Multiple User (MAMU) service nodes. Initial setup and configuration of MAMU nodes can be done during CLE installation or upgrades using the CLE installer. These initial steps can also be done manually without using the installer. For more information, see *Installing and Configuring Cray Linux Environment (CLE) Software* (S–2444).

13.1 About Service Node MAMU

Service Node MAMU support provides the ability to set aside a small number of repurposed compute nodes for serial workload. These nodes serve as workload management execution nodes, specifically designated for serial workload, and intra-node MPI. The workload manager manages these as standard Linux nodes, and support core level placement. Users place jobs directly to the nodes by submitting their jobs to a designated workload manager queue. These serial workload nodes do not run ALPS, and thus cannot be used to place jobs to the compute nodes.

Administrators have the ability to resize the size of the serial workload cluster without a full-system reboot. Nodes must be preconfigured to serve as serial workload nodes, but once this configuration is in place, nodes can be moved back and forth between the serial workload queues and ALPS managed Cray compute nodes simply by rebooting the nodes.

Service Node MAMU also provides `cgroup`-based out of memory protection for the serial workload nodes.

13.1.1 Limitations

1. Currently, this feature is only qualified with the PBSPro workload manager.
2. PBS 12.1 or later is required.
3. Users may only submit jobs to a single node. If the `cgroup_hook.py` is in place, the hook will reject any job that attempts to span multiple nodes as invalid.
4. A system can have no more than 100 serial workload nodes.
5. Cray-specific options (core affinity, node health checking, and so on) are only supported to the extent that the workload manager vendor supports these features.

6. The serial workload nodes do not support `ssh` login (except for `root` and `crayadm` administrative users) to guarantee out-of-memory protection.

7. Sites using power capping may need to take additional action when repurposing nodes. For more information, see *Monitoring and Managing Power Consumption on the Cray XC30 System* (S–0043).

13.2 Manually Configuring MAMU Nodes

Use the steps in this section to enable service Node MAMU without running the CLE installer. Also, use these steps to add additional MAMU nodes without running the installer. This is an alternative method; the installer can also be rerun to configure new nodes.

Procedure 104. Configuring MAMU node manually

1. Log on to the boot node as `root`.

   ```
   crayadm@smw$ ssh boot -l root
   ```

2. Run `xtopview` to configure a new class, `postproc` in this example, using the template for service class. Do this step once.

   ```
   boot# xtopview
   default/:/# xtcloneshared -o -c service postproc
   ```

3. For each new MAMU node, create a node-specialized view in the class `postproc`. **This command overwrites any existing node specialization for the given NID.** The assumption is that MAMU nodes are repurposed compute nodes. Do not attempt this procedure on service nodes. In the following example, a view for a node identified by `nid00039` is created.

   ```
   default/:/# xtcloneshared -o -N postproc 39
   default/:/# exit
   ```

4. Configure services on the postprocessing nodes in the `postproc` class:

   ```
   boot:# xtopview -c postproc -x /etc/opt/cray/sdb/node_classes
   postproc/:/# touch /etc/sysctl.conf
   postproc/:/# xtspec /etc/sysctl.conf
   postproc/:/# echo "vm.oom_appkill=1" >> /etc/sysctl.conf
   postproc/:/# chkconfig nhcdbd_script off
   postproc/:/# chkconfig node_health off
   postproc/:/# chkconfig alps off
   postproc/:/# chkconfig xpmem on
   postproc/:/# xtspec /etc/ssh/sshd_config
   postproc/:/# vi /etc/ssh/sshd_config
   ```

   Remove any `MatchUser` blocks from the end of the file.

   ```
   postproc/:/# echo "AllowUsers root crayadm" >>
   /etc/ssh/sshd_config
   postproc/:/# exit
   ```

5. Configure node classes.

```
boot:# xtopview -x /etc/opt/cray/sdb/node_classes
default/:/# touch /etc/opt/cray/sdb/class_roles
default/:/# xtspec /etc/opt/cray/sdb/class_roles
default/:/# touch /etc/opt/cray/sdb/class_roles
default/:/# cp /etc/opt/cray/sdb/class_roles /etc/opt/cray/sdb/class_roles.sav
default/:/# exit
boot:# xtopview -x /etc/opt/cray/sdb/node_classes
default/:/# sed -i '/\( \t\)*postproc\( \t\)*:.*/d' /etc/opt/cray/sdb/class_roles; \
            echo postproc:service_mamu >>     /etc/opt/cray/sdb/class_roles; \
            cat /etc/opt/cray/sdb/class_roles
default/:/# exit
```

6. Fix `node_classes` and services.

```
boot# echo "39:postproc" >> /etc/opt/cray/sdb/node_classes
```

7. Edit `/var/opt/cray/install/shell_ssh.sh` on the boot node, and add an entry for the NID and cname prior to the copy of `known_hosts`.

```
do_ssh nid00039
do_ssh c0-0c0s9n3
```

8. Add cname entries to the `pdsh machines` file. This is necessary for `xtshutdown` to halt the nodes correctly.

```
boot:# echo "c0-0c0s9n3" >> /etc/opt/cray/pdsh/machines
```

9. Run `shell_ssh.sh` to update the known hosts file with the updated keys and make them available on the nodes.

```
boot:# cd /var/opt/cray/install
boot:# mv /root/.ssh/known_hosts  /root/.ssh/known_hosts.save
boot:# ./shell_ssh.sh
```

10. Create persistent `/var` in mountpoint. Typically, this is `/snv` on the boot node.

```
boot# mkdir -p /snv/39
boot# cd /snv/39
boot# tar -zkxf /rr/current/.shared/var-skel.tgz
boot# chown root:root /snv/39/var
boot# chmod 755 /snv/39/var
boot# chmod 1755 /snv/39/var/tmp
boot# mkdir -p /snv/39/var/lib/ntp/drift
boot# chown ntp /snv/39/var/lib/ntp/drift
```

13.3 Configuring MAMU Nodes in PBS Pro

The recommended configuration below assume that PBS is already configured and running on the Cray mainframe. This section outlines the necessary steps to configure one PBS complex to manage multiple architectures.

The configuration below uses the Cray XT and Linux architectures as examples. The Linux architecture can be an external or internal service node, or a repurposed Cray compute node booted with a service image.

The configuration used in this section will not work with `arch` settings on your queues. The Hook assumes all jobs are Cray XT jobs unless you specify otherwise.

13.3.1 Adding a Non-Cray XT MOM to the Complex

A non-Cray XT MOM is a MAMU or post processing node (ppn) node. Repeat the procedure for each added node.

Procedure 105. Adding a Non-Cray XT MOM

1. Create a custom tag to differentiate the Linux nodes from the Cray compute nodes. Add this boolean custom resource to the PBS `resourcedef` file on the PBS server node.

   ```
   login# vi /var/spool/PBS/server_priv/resourcedef
   ```

   Add this line.

   ```
   ppn  type=boolean  flag=h
   ```

2. Ensure the hostnames are setup correctly on the Linux hosts. Use hostnames that describe the function of these nodes. The hostname should be present in `/etc/hosts` as well as what is returned by the `hostname` command.

   ```
   login# cat /etc/hosts | grep ppn
   10.128.0.31    nid00030 c0-0c0s7n2 ppn30
   ppn30:~ # hostname
   ppn30
   ```

3. If the hostname is not set correctly, do these steps to change it.

   ```
   boot# xtopview -n 30
   node/30:/ # xtspec -n 30 /etc/HOSTNAME
   node/30:/ # vi /etc/HOSTNAME
   ```

   Add this string.

   ```
   ppn30
   ```

   ```
   node/30:/ # exit
   smw> xtbootsys --reboot -L SNL0 c0-0c0s7n2
   ```

4. Create the node inside of PBS. Do this step for each node you are adding to your complex.

   ```
   login# qmgr -c "create node ppn30"
   ```

5. Start the PBS MOM daemon on each node.

   ```
   ppn30:~ # ppn30:~ # /etc/init.d/pbs start
   ```

Automate this by using the pdsh command.

```
boot# pdsh -w ppn[1-N] "/etc/init.d/pbs start"
```

6. Configure PBS MOM configuration file $usecp variables. **Do not set alps_client.**

```
ppn30:/var/spool/PBS/mom_priv # cat config
$usecp *:/home /home
$usecp *:/ufs /ufs
$usecp *:/cray /cray
```

7. Verify that the node is configured in PBS and state is free.

```
login# qmgr -c "print node ppn30"
#
# Create nodes and set their properties.
#
#
# Create and define node ppn30
#
create node ppn30 Mom=nid00030
set node ppn30 state = free
set node ppn30 resources_available.arch = linux
set node ppn30 resources_available.host = nid00030
set node ppn30 resources_available.mem = 32993312kb
set node ppn30 resources_available.ncpus = 16
set node ppn30 resources_available.vnode = ppn30
set node ppn30 resv_enable = True
set node ppn30 sharing = default_shared
```

8. Tag each node in the complex with ppn=false.

```
login# qmgr -c "set node @default resources_available.ppn = false"
```

9. Now tag each new Linux node with ppn=true.

```
login# qmgr -c "set node ppn30 resources_available.ppn = true"
```

10. Verify this new resource is present.

```
login# qmgr -c "print node ppn30"
#
# Create nodes and set their properties.
#
#
# Create and define node ppn30
#
create node ppn30 Mom=nid00030
set node ppn30 state = free
set node ppn30 resources_available.arch = linux
set node ppn30 resources_available.host = nid00030
set node ppn30 resources_available.mem = 32993312kb
set node ppn30 resources_available.ncpus = 16
set node ppn30 resources_available.ppn = True
set node ppn30 resources_available.vnode = ppn30
set node ppn30 resv_enable = True
set node ppn30 sharing = default_shared
```

Also verify that all Cray XT nodes have `ppn=false`.

11. Run jobs targeting the ppn nodes.

```
login# qsub -I -lselect=1:ppn=true
qsub: waiting for job 61.login to start
qsub: job 61.login ready

ppn30:~ #
```

12. Note that if there is not a default memory setting for the MAMU or ppn nodes, specify the expected maximum amount of memory to use.

```
admin@login:~>qsub -I -lselect=1:ppn=true:mem=200mb
qsub: waiting for job 6114.login to start
qsub: job 6114.login ready

crayadm@ppn30:~>
```

13.3.2 Setting Up the PBS Scheduler

Edit the file `/var/spool/PBS/sched_priv/sched_config` to set the scheduling policy `smp_cluster_dist` to `smp_cluster_dist:round_robin`.

13.3.3 Enabling the `cgroup` Hook

Cray supplies a `cgroup` hook for use with PBS Pro. This hook makes use of the Linux kernel `cgroup` feature to enforce PBS job limits for memory and number of CPUs. The hook creates a memory `cgroup` for each job to enforce the memory limit, thus all jobs must have a memory limit assigned either by the user or by PBS. The hook creates a `cpuset` for each job to enforce CPU allocations, thus all jobs must have a number of CPUs assigned either by the user or by PBS.

Because the hook creates `cpuset`s for each job, use of the hook also enables the use of the Cray OOM application killer feature which kills all processes within a job whenever any one process in the job is chosen to be killed due to an out-of-memory (OOM) condition. The hook does affect any jobs run on Cray login or Cray compute nodes, meaning, those jobs with a PBS `vntype` of `cray_login` or `cray_compute`.

The `cgroup` hook is installed in PBS Pro using the `qmgr` command to:

- Create the hook
- Define the events for the hook
- Import the hook code

For more information about managing hooks in PBS, see the *PBS Professional Administrator's Guide*.

Procedure 106. Enabling the cgroup hook

1. Create the hook and set the events using the `create hook` command.

   ```
   # qmgr -c 'create hook cray_cgroup \
   event="execjob_prologue,execjob_end"'
   ```

2. Import the hook code into PBS each time new or upgraded hook code is to be used.

   ```
   # qmgr -c 'import hook cray_cgroup application/x-python \
   default /opt/cray/pbs_hooks/default/src/cray_hooks/cgroup_hook.py'
   ```

3. Alternatively, import a specific version of the hook code by replacing default with the specific version, as in this example.

```
# qmgr -c 'import hook cray_cgroup application/x-python \
default /opt/cray/pbs_hooks/1.0.0-1.0000.47942.0.0.ari/src/cray_hooks/main/cgroup.py'
```

13.3.4 Creating and Importing Prologue and Epilogue Hooks

Enabling a hook to use the `execjob_prologue` event disables any prologue bash scripts in PBS. Likewise, enabling a hook to use the `execjob_epilogue` event disables any epilogue bash scripts in PBS.

Creating and importing prologue and epilogue hooks are important for getting CCM working on the system.

The following prologue and epilogue hooks are wrappers for the prologue/epilogue scripts. Installed them on any system that needs both hook and prologue/epilogue script functionality.

Prologue hooks

```
qmgr -c 'create hook cray_prologue'
qmgr -c 'set hook cray_prologue type = site'
qmgr -c 'set hook cray_prologue enabled = true'
qmgr -c 'set hook cray_prologue event = execjob_prologue'
qmgr -c 'set hook cray_prologue user = pbsadmin'
qmgr -c 'set hook cray_prologue alarm = 30'
qmgr -c 'set hook cray_prologue order = 1'
```

Epilogue hooks

```
qmgr -c 'create hook cray_epilogue'
qmgr -c 'set hook cray_epilogue type = site'
qmgr -c 'set hook cray_epilogue enabled = true'
qmgr -c 'set hook cray_epilogue event = execjob_epilogue'
qmgr -c 'set hook cray_epilogue user = pbsadmin'
qmgr -c 'set hook cray_epilogue alarm = 30'
qmgr -c 'set hook cray_epilogue order = 1'
```

Procedure 107. Import prologue hooks

1. Create the following temporary file.

   ```
   login:~ # vi /var/spool/PBS/mom_priv/prologue.py
   ```

2. Copy and past the following text into `prologue.py`.

```
"""
Copyright 2013 Cray Inc. All rights reserved.
This script sets up the environment for the main hook code then invokes the
main routine from the library on disk where this hook was installed.  One
benefit of this organization is to improve the testability of the main routine
by allowing it to work, without modification, under either PBS or a test
environment.
Description:
This hook will execute a prologue script located at ${PBS_HOME}/mom_priv/prologue
if the file exists, and there are arguments to pass.
"""
import pbs
import sys
import os
# For now the PBS_HOME value has to be editeded manually
# There is no way to access PBS_HOME with the pbs module
PBS_HOME = '/var/spool/PBS-12.1.400/'
PROLOGUE_DIR = PBS_HOME + '/mom_priv'
PROLOGUE = PROLOGUE_DIR + '/prologue'
try:
    # Get the hook event information and parameters
    e = pbs.event()
    # Check to see if a prologe even exists
    if os.path.exists(PROLOGUE) == False:
        e.accept()
    # Ignore requests from scheduler or server
    if e.requestor in ["PBS_Server", "Scheduler"]:
        e.accept()
    # Get the information for the job being queued
    j = e.job
    if j and j.id and j.euser and j.egroup:
        # Assemble and execute the prolgue command
        cmd = PROLOGUE
        cmd = cmd + ' ' + j.id
        cmd = cmd + ' ' + j.euser
        cmd = cmd + ' ' + j.egroup
        os.popen(cmd, 'w')
    # accept the event
    e.accept()
except SystemExit:
    pass
except:
    print sys.exc_info()[0]
```

3. Import the prologue.

   ```
   login:~ # qmgr -c 'import hook cray_prologue \
   application/x-python default /var/spool/PBS/mom_priv/prologue.py'
   ```

Procedure 108. Importing epilogue hooks

1. Create the following temporary file.

```
login:~ # vi /var/spool/PBSmom_priv/epilogue.py
```

2. Copy and past the following text into `epilogue.py`.

```
"""
Copyright 2013 Cray Inc. All rights reserved.
This script sets up the environment for the main hook code then invokes the
main routine from the library on disk where this hook was installed.  One
benefit of this organization is to improve the testability of the main routine
by allowing it to work, without modification, under either PBS or a test
environment.
Description:
This hook will execute a epilogue script located at ${PBS_HOME}/mom_priv/epilogue
if the file exists, and there are 3 arguments to pass. Passing more than 3
arguments is unsupported
"""
import pbs
import sys
import os
# The PBS_HOME value has to be editeded manually
# There is no way to access PBS_HOME with the pbs module
PBS_HOME = '/var/spool/PBS-12.1.400/'
EPILOGUE_DIR = PBS_HOME + '/mom_priv'
EPILOGUE = EPILOGUE_DIR + '/epilogue'
try:
    # Get the hook event information and parameters
    e = pbs.event()
        if os.path.exists(EPILOGUE) == False:
            e.accept()
    # Ignore requests from scheduler or server
    if e.requestor in ["PBS_Server", "Scheduler"]:
        e.accept()
    # Get the information for the job being queued
    j = e.job
    if j and j.id and j.euser and j.egroup:
        # Assemble and execute the epilogue command
        cmd = EPILOGUE
        cmd = cmd + ' ' + j.id
        cmd = cmd + ' ' + j.euser
        cmd = cmd + ' ' + j.egroup
        os.popen(cmd, 'w')
    # accept the event
    e.accept()
except SystemExit:
    pass
except:
    print sys.exc_info()[0]
```

3. Import the epilogue.

```
login:~ # qmgr -c 'import hook cray_epilogue \
application/x-python default /var/spool/pbs/mom_priv/epilogue.py'
```

13.4 Configuring Network Connectivity for User Account Management

To connect to an external network from the serial workload nodes, those nodes must have external network connectivity. The recommendation is to provide this connectivity through a gateway route to a host that requires network connectivity.

To use LDAP to an external host, the MAMU nodes must be configured to have network connectivity. The LDAP configuration itself is the same as for any internal login node. What is different is that repurposed nodes do not have direct Ethernet connectivity. One approach is to use the boot automation script to run this command on each MAMU node, to specify a gateway node that does have an Ethernet connection:

```
/sbin/route add default gw IP_addresses
```

SMW and CLE System Administration Commands [A]

In addition to the SUSE Linux Enterprise Server (SLES) commands available to you, this appendix lists the Cray developed commands for administering CLE on your Cray system.

The system provides the following types of commands for the system administrator:

- Hardware Supervisory System (HSS) commands invoked from the System Management Workstation (SMW) to control HSS operations; HSS commands are provided with SMW release packages.
- Cray Lightweight Log Management (LLM) System commands invoked from the SMW or on a CLE service node; the LLM commands are provided with both the SMW release packages and the CLE release packages.
- Cray Linux Environment (CLE) commands invoked from a node to control the service and compute partitions; CLE commands are provided with CLE release packages.

A.1 HSS Commands

Table 12 shows the HSS commands and their functions.

Table 12. HSS Commands

Command	Description
`dbMonitor`	Controls the monitor process script that starts during system boot to watch `mysqld` and restart `mysqld` if it should crash
`getSedcLogValues`	Displays specified `sedc_manager` log file records
`hss_make_default_initrd`	Creates the default HSS controller boot image
`hssbootlink`	Links a Linux kernel `bzImage` file, an `initramfs` file, and a `parameters` file so that they can be booted on a Controller by using PXE boot on an SMW
`hssclone`	Clones the master image directory

Command	Description
`hssds_init`	Creates the Hardware Supervisory System (HSS) data store; ensures the proper HSS data store user credentials are created and that the data store is ready for operation
`hsspackage`	Facilitates creation of controller boot images
`nid2nic`	Prints all *nid*-to-*nic* address mappings
`rtr`	Routes the Cray network
`sedc_manager`	Invokes the System Environment Data Collections (SEDC) SMW manager
`SMWconfig`	Automatically configures software on SMW
`SMWinstall`	Automatically installs and configures software on SMW
`SMWinstallCLE`	Updates the SMW software on `bootroot` and `sharedroot` for system sets with CLE software installed
`xtalive`	Gets a response from an HSS daemon
`xtbootdump`	Parses a *bootinfo-file* to determine if `xtdumpsys` needs to be invoked
`xtbootimg`	Creates, extracts, or updates a Cray bootable image file
`xtbootsys`	Boots specified components in a Cray system
`xtbounce`	Powers components of the Cray system down then up
`xtccreboot`	Reboots specified cabinet or blade controllers
`xtcheckhss`	Initiates a series of tests that validate the health of the HSS
`xtcheckmac`	Checks for duplicate MAC addresses among L1 and L0 controllers
`xtclass`	Displays the network topology class for this system
`xtclear`	Clears component flags in the state manager
`xtclear_link_alerts`	Clears alert flags
`xtcli`	Runs the HSS command line
`xtcli boot`	Specifies the types of components to boot
`xtcli clear`	Clears flag status in component state
`xtcli part`	Updates partition configurations
`xtcli power`	Powers a component up or down
`xtcli set`	Sets flag status in the component state
`xtcon`	Provides a two-way connection to the console of any running service node
`xtconsole`	Displays console text from a node
`xtconsumer`	Displays HSS events

Command	Description
xtcpreport	Parses xtnlrd log file and display system network congestion protection information
xtcptop	Parses specified xtnlrd log file and displays real-time system network congestion protection information as it is written to the file
xtdiscover	Discovers and configures the Cray system hardware
xtdumpsys	Gathers information when a system stops responding or fails
xterrorcode	Displays event error codes
xtfileio	Reads or writes a file on an L1 or L0 controller
xtgenid	Generates HSS physical IDs
xthb	Node heartbeat checker
xthwerrlog	Reports hardware errors
xthwerrlogd	Logs Gemini network errors
xthwinv	Retrieves hardware component information for selected modules
xtlogfilter	Filters information from event router log files
xtlogin	Logs on to cabinet and blade control processors
xtmcinfo	Gets microcontroller information from cabinet and blade control processors
xtmem2file	Reads CPU or Cray Gemini memory and saves it in a file
xtmemio	Reads or writes 32-bit or 64-bit words from CPU or Cray Gemini memory
xtnetwatch	Watches the Cray system interconnection network for link control block (LCB) and router errors
xtnid2str	Converts node identification numbers to physical names
xtnlrd	Responds to fatal link errors by rerouting the system
xtnmi	Collects debug information from unresponsive nodes
xtpcimon	Monitors health of PCIe channels
xtpget	Displays current system power usage and applied capping parameters.
xtpmaction	Implements power management actions.
xtpmdbconfig	Modifies power management configuration parameters and provides a mechanism for an operator to hook into database rotation events.
xtpscan	Controls power data collection and logging.
xtresview	Displays the current state of cabinets, blades, and links, and whether any have failed or been warm swapped out
xtrsh	Invokes a diagnostic utility that concurrently executes programs on batches of cabinet control processors and/or blade control processors

Command	Description
`xtsedcviewer`	Command-line interface for SEDC
`xtshow`	Shows components with selected characteristics
`xtwarmswap`	Allows Cray system blades, chassis, or cabinets to be warm swapped
`xtwatchsyslog`	Shows all log messages for cabinet control processors (L1 controllers) and blade control processors (L0 controllers)

A.2 Cray Lightweight Log Management (LLM) System Commands

Table 13 shows the LLM commands and their functions.

Table 13. LLM Commands

Command	Description
`cray-syslog`	Starts, stops, restarts, or checks the status of the log system
`xtlog`	Delivers messages to the Cray Lightweight Log Management (LLM) system
`xtsession`	Displays the current boot *sessionid*
`xttail`	Outputs the last part of Cray Lightweight Log Management (LLM) files
`xttoday`	Provides today's date in same format that is used to time stamp Cray Lightweight Log Management (LLM) log files
`xttrim`	Provides a simple and configurable method to automate the compression and deletion of old log files

A.3 CLE System Administration Commands

Table 14 shows CLE commands and their functions.

Table 14. CLE Commands

Command	Description
`apmgr`	Provides interface for ALPS to cancel pending reservations.
`apconf`	A utility for manipulating and modifying ALPS configuration files.
`cdump`	Dumps node memory.
`clcvt`	A utility for configuring and validating Fine-grained Routing (FGR) on Cray systems.
`crash`	Analyzes Linux crash dump data or a live system (Red Hat utility).
`csacon`	Condenses records from the sorted `pacct` file.

Command	Description
csanodeacct	Initiates the end of application accounting on a node.
csanodemerg	Initiates collection of individual compute node accounting files.
csanodesum	Reads and consolidates application node accounting records.
dumpd	Initiates automatic dump and reboot of nodes when requested by Node Health Checker (NHC).
lastlogin	Records last date on which each user logged in
lbcd	Invokes the load balancer client daemon.
lbnamed	Invokes the load balancer service daemon.
lustre_control	Manages direct-attached and external Lustre file systems using standard Lustre commands and site specific Lustre file system definition and tuning files.
nhc_recovery	Releases compute nodes on a crashed login node that will not be rebooted.
pdsh	Issues commands to groups of hosts in parallel.
projdb	Creates and updates system project database for CSA.
rca-helper	Used in various administrative scripts to retrieve information from the Resiliency Communication Agent (RCA).
rsipd	Invokes the Realm-Specific IP Gateway Server.
sdbwarmswap	Updates the Service Database (SDB) when blades are replaced or added.
xt-lustre-proxy	Invokes the Lustre startup/shutdown, health monitor, and automatic failover utility.
xtalloc2db	Converts a text file to the alloc_mode table in the Service Database (SDB).
xtattr2db	Converts a text file to the attributes table in the Service Database (SDB).
xtauditctl	Distributes auditctl requests to nodes on a Cray system.
xtaumerge	Merges audit logs from multiple nodes into a single audit log file.
xtcdr2proc	Gets information from the RCA.
xtcheckhealth	Executes the Node Health Checker.
xtcleanup_after	Called by ALPS to check node health.
xtclone	Clones the master image directory and overlays a site-specific template.
xtcloneshared	Clones node or class directory in shared root hierarchy.
xtdb2alloc	Converts the alloc_mode table in the Service Database (SDB) to a text file.

Command	Description
xtdb2attr	Converts the attributes table in the Service Database (SDB) to a text file.
xtdb2etchosts	Converts service information in the SDB to a text file.
xtdb2filesys	Converts the filesystem table of the SDB to a text file.
xtdb2gpus	Converts the gpus table in the Service Database (SDB) to a text file.
xtdb2lustrefailover	Converts the lustre_failover table in the SDB to a text file.
xtdb2lustreserv	Converts the lustre_serv table of the SDB to a text file.
xtdb2nodeclasses	Converts the service_processor table of the SDB to a text file.
xtdb2order	Converts the processor table od_allocator_id field in the Service Database (SDB) to a text file.
xtdb2proc	Converts the processor table of the SDB to a text file.
xtdb2segment	Converts segment table in the Service Database (SDB) to a text file.
xtdb2servcmd	Converts the service_cmd table of the SDB to a text file.
xtdb2servconfig	Converts the service_config table of the SDB to a text file.
xtdbsyncd	Invokes the HSS/SDB synchronization daemon.
xtfilesys2db	Converts a text file to the SDB filesystem table.
xtfsck	Checks file systems on a system set defined in /etc/sysset.conf.
xtgetconfig	Gets configuration information from /etc/sysconfig/xt file.
xtgetdslroot	Returns compute node root path used within an environment.
xtgpus2db	Converts a text file to the SDB gpus table.
xthotbackup	Creates a backup copy of a system set on the boot RAID.
xthowspec	Displays file specialization in the shared root directory.
xtlusfoevntsndr	Sends failover events to clients for Lustre imperative recovery.
xtlusfoadmin	Displays Lustre automatic failover database tables and enables/disables Lustre server failover.
xtlustrefailover2db	Converts a text file to the SDB lustre_failover table.
xtlustreserv2db	Converts a text file to the SDB lustre_service table.
xtmount	Allows administrators to mount storage devices on the SMW based on their LABEL and FUNCTION roles in the sysset.conf file instead of long /dev/disk/by-id names.
xtnce	Displays or changes the class of a node.
xtnodeclasses2db	Converts a text file to the service_processor table in the SDB.
xtnodestat	Provides current job and node status summary information on a CNL compute node.

Command	Description
`xtoparchive`	Performs archive operations on shared root files from a given specification list.
`xtopco`	Checks out RCS versioned shared root specialized files.
`xtopcommit`	Commits changes made inside an `xtopview` session.
`xtoprdump`	Lists shared root file specification and version information.
`xtoprlog`	Provides RCS log information about shared root specialized files.
`xtopview`	Views file system as it would appear on any node, class of nodes, or all service nodes.
`xtorder2db`	Converts a text file to values in the `od_allocator_id` field of the processor table in the Service Database (SDB).
`xtpackage`	Facilitates creation of boot images.
`xtpkgvar`	Creates a skeleton structure of `/var`.
`xtproc2db`	Converts a text file to the `processor` table of the SDB.
`xtprocadmin`	Gets/sets the processor flag in the SDB.
`xtrelswitch`	Performs release switching by manipulating symbolic links in the file system and by setting the default version of modulefiles that are loaded at login.
`xtrsipcfg`	Generates and optionally installs the necessary RSIP client and server configuration files.
`xtsegment2db`	Converts a text file to `segment` table in the Service Database (SDB).
`xtservcmd2db`	Converts a text file to the `service_cmd` table of the SDB.
`xtservconfig`	Adds, removes, or modifies the `service_config` table of the SDB.
`xtservconfig2db`	Converts a text file to the `service_config` table of the SDB.
`xtshutdown`	Shuts down the service nodes in an orderly fashion.
`xtspec`	Specializes files for nodes or classes.
`xtunspec`	Unspecializes files for nodes or classes.
`xtverifyconfig`	Verifies the coherency of `/etc/init.d` files across all shared root views.
`xtverifydefaults`	Verifies and fixes inconsistent system default links within the shared root.
`xtverifyshroot`	Checks the configuration of the shared-root file system.

System States [B]

Table 15 defines state definitions for system components. States are designated by uppercase letters. Table 16 shows states that are common to all components.

Note: The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If you disable a component, the state shown becomes `disabled`. When you use the `xtcli enable` command to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

The state of `EMPTY` components does not change when using the `xtcli enable` or the `xtcli disable` command, unless the force option (`-f`) is used.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Table 15. State Definitions

State	Cabinet Controller	Blade Controller	Cray ASIC	CPU	Link
`OFF`	Powered off	Powered off	Powered off	Powered off	Link is down
`ON`	Powered on	Powered on	Powered on and operational	Powered on	Link is up
`HALT`	–	–	–	CPU halted	–
`STANDBY`	–	–	–	Booting was initiated	–
`READY`	Operational	Operational	Operational	Booted	Operational

Table 16. Additional State Definitions

State	Description
DISABLED	Operator disabled this component.
EMPTY	Component does not exist.
N/A	Component cannot be accessed by the system.
RESVD	Reserved; new jobs are not allocated to this component.

There are two notification flags, which can occur with any state.

- WARNING

  A condition of the component was detected that is outside the normal operating range but is not yet dangerous.

- ALERT

  A dangerous condition or fatal error has been detected for the component.

Table 17 shows the states by component for which the `xtcli` commands run.

Note: Administrative states are hierarchal, so disabling or enabling a component has a cascading effect on that component's children. A component may not be enabled if its parent component is disabled, but a subcomponent may be disabled without affecting its parents.

Table 17. `xtcli` Commands and Allowed States

`xtcli` Command	Subcommand	Cabinet Controller	Blade Controller	Node
power	up	ON	OFF	OFF
	down	READY	ON	ON, HALT, DIAG
	up_slot (an alias for up)			
	down_slot (an alias for down)			
	force_down (an alias for down)			
halt		N/A	N/A	STANDBY, READY
boot		N/A	N/A	ON, HALT

Remote Access to the SMW [C]

Virtual Network Computing (VNC) software enables you to view and interact with the SMW from another computer. The Cray system provides a VNC server, `Xvnc`. You must download a VNC client to connect to it. See RealVNC (http://www.realvnc.com/) or TightVNC (http://www.tightvnc.com/) for more information.

Note: The VNC software requires a TCP/IP connection between the server and the viewer. Some firewalls may not allow this connection due to security policies. VNC is considered to be an insecure protocol. See Procedure 111 on page 397 for a procedure to create a secure connection to the SMW.

Cray supplies a VNC account `cray-vnc`.

Procedure 109. Starting the VNC server

1. Log on to the SMW as `root` user.
2. Use the `chkconfig` command to check the current status of the server:

   ```
   smw:~ # chkconfig vnc
   vnc  off
   ```

3. Disable `xinetd` startup of `Xvnc`.

   If the `chkconfig` command you executed in step 2 reports that `Xvnc` was started by INET services (`xinetd`):

   ```
   smw:~ # chkconfig vnc
   vnc xinetd
   ```

   Execute the following commands to disable `xinetd` startup of `Xvnc` (`xinetd` startup of `Xvnc` is the SLES 11 default, but it usually is disabled by `chkconfig`):

   ```
   smw:~ # chkconfig vnc off
   smw:~ # /etc/init.d/xinetd reload
   Reload INET services (xinetd).                    done
   ```

   If no other `xinetd` services have been enabled, the `reload` command will return `failed` instead of `done`. If the `reload` command returns `failed`, this is normal and you can ignore the `failed` notification.

4. Use the `chkconfig` command to start `Xvnc` at boot time:

   ```
   smw:~ # chkconfig vnc on
   ```

5. Start the Xvnc server immediately:

```
smw:~ # /etc/init.d/vnc start
```

   If the password for cray-vnc has not already been established, the system prompts you for one. You must enter a password to access the server.

```
Password: ********
Verify:
Would you like to enter a view-only password (y/n)? n
xauth:  creating new authority file /home/cray-vnc/.Xauthority

New 'X' desktop is smw:1

Creating default startup script /home/cray-vnc/.vnc/xstartup
Starting applications specified in /home/cray-vnc/.vnc/xstartup
Log file is /home/cray-vnc/.vnc/smw:1.log
```

   To access the Xvnc server, use a VNC client, such as vncviewer, tight_VNC, vnc4, or a web browser. Direct it to the SMW that is running Xvnc. Many clients allow you to specify whether you want to connect in view-only or in an active mode. If you choose active participation, every mouse movement and keystroke made in your client is sent to the server. If more than one client is active at the same time, your typing and mouse movements are intermixed.

   Commands entered through the VNC client affect the system as if they were entered from the SMW. However, the main SMW window and the VNC clients cannot detect each other. It is a good idea for the administrator who is sitting at the SMW to access the system through a VNC client.

   The startup script starts the Xvnc server for display :1.

6. Verify that Xvnc started:

```
smw:~ # ps -e | grep vnc
1839 pts/0    00:00:00 Xvnc
```

Procedure 110. For workstation or laptop running Linux: Connecting to the VNC server through an SSH tunnel, using the vncviewer -via option

Important: This procedure is for use with the TightVNC client program.

Verify that you have the vncviewer -via option available. If you do not, use Procedure 111 on page 397.

- If you are connecting from a workstation or laptop running Linux, enter the vncviewer command shown below.

The first password you enter is for `crayadm` on the SMW. The second password you enter is for the VNC server on the SMW, which was set when the VNC server was started for the first time using `/etc/init.d/vnc start` on the SMW.

```
/home/mary> vncviewer -via crayadm@smw localhost:1
Password: ********
VNC server supports protocol version 3.130 (viewer 3.3)
Password: ********
VNC authentication succeeded
Desktop name "cray-vnc's X desktop (smw:1)"
Connected to VNC server, using protocol version 3.3
```

Procedure 111. For workstation or laptop running Linux: Connecting to the VNC server through an SSH tunnel

Note: This procedure assumes that the VNC server on the SMW is running with the default port of `5901`.

1. This `ssh` command starts an `ssh` session between the local Linux computer and the SMW, and it also creates an SSH tunnel so that port `5902` on the *localhost* is forwarded through the encrypted SSH tunnel to port `5901` on the SMW. You will be prompted for the `crayadm` password on the SMW.

   ```
   local_linux_prompt> ssh -L 5902:localhost:5901 smw -l crayadm
   Password:
   crayadm@smw>
   ```

2. Now `vncviewer` can be started using the local side of the SSH tunnel, which is port `5902`. You will be prompted for the password of the VNC server on the SMW. This password was set when the VNC server was started for the first time using `/etc/init.d/vnc start` on the SMW.

   ```
   local_linux_prompt> vncviewer localhost:2
   Connected to RFB server, using protocol version 3.7
   Performing standard VNC authentication
   Password:
   ```

   The VNC window from the SMW appears. All traffic between the `vncviewer` on the local Linux computer and the VNC server on the SMW is now encrypted through the SSH tunnel.

Procedure 112. For workstation or laptop running Mac OS X: Connecting to the VNC server through an SSH tunnel

Note: This procedure assumes that the VNC server on the SMW is running with the default port of `5901`.

1. This `ssh` command starts an `ssh` session between the local Mac OS X computer and the SMW, and it also creates an SSH tunnel so that port `5902` on the localhost is forwarded through the encrypted SSH tunnel to port `5901` on the SMW. You will be prompted for the `crayadm` password on the SMW.

   ```
   local_mac_prompt> ssh -L 5902:localhost:5901 smw -l crayadm
   Password:
   crayadm@smw>
   ```

2. Now `vncviewer` can be started using the local side of the SSH tunnel, which is port `5902`. You will be prompted for the password of the VNC server on the SMW. This password was set when the VNC server was started for the first time using `/etc/init.d/vnc start` on the SMW.

   If you type this on the Mac OS X command line after having prepared the SSH tunnel, the `vncviewer` will pop up:

   ```
   local_mac_prompt% open vnc://localhost:5902
   ```

   The VNC window from the SMW appears. All traffic between the `vncviewer` on the local Mac OS X computer and the VNC server on the SMW is now encrypted through the SSH tunnel.

Procedure 113. For workstation or laptop running Windows: Connecting to the VNC server through an SSH tunnel

Note: If you are connecting from a computer running Windows, then both a VNC client program, such as TightVNC and an SSH program, such as PuTTY, SecureCRT, or OpenSSH are recommended.

1. The same method described in Procedure 111 can be used for computers running the Windows operating system.

   Although TightVNC encrypts VNC passwords sent over the network, the rest of the traffic is sent unencrypted. To avoid a security risk, install and configure an SSH program that creates an SSH tunnel between TightVNC on the local computer (localhost port `5902`) and the remote VNC server (localhost port `5901`).

   Note: Details about how to create the SSH tunnel vary amongst the different SSH programs for Windows computers.

2. After installing TightVNC, start the VNC viewer program by double-clicking on the **TightVNC** icon. Enter the host name and VNC screen number, `localhost:`*number* (such as, `localhost:2` or `localhost:5902`), and then select the **Connect** button.

Updating the Time Zone [D]

When you install the Cray Linux Environment (CLE) operating system, the Cray system time is set at US/Central Standard Time (CST), which is six hours behind Greenwich Mean Time (GMT). You can change this time.

Note: When a Cray system is initially installed, the time zone set on the SMW is copied to the boot root, shared root and CNL boot images.

To change the time zone on the SMW, L0 controller, L1 controller, boot root, shared root, or for the compute node image, follow the appropriate procedure below.

Procedure 114. Changing the time zone for the SMW and the blade and cabinet controllers

Caution: Perform this procedure while the Cray system is shut down; do not flash blade and cabinet controllers while the Cray system is booted.

You must be logged on as `root`. In this example, the time zone is changed from `"America/Chicago"` to `"America/New_York"`.

1. Ensure the blade and cabinet controllers are responding. For example:

   ```
   smw:~ # xtalive -a l0sysd s0
   ```

2. Check the current time zone setting for the SMW and controllers.

   ```
   smw:~ # date
   Wed Aug 01 21:30:06 CDT 2012

   smw:~ # xtrsh -l root -s /bin/date s0
   c0-0c0s2 : Wed Aug 01 21:30:51 CDT 2012
   c0-0c0s5 : Wed Aug 01 21:30:51 CDT 2012
   c0-0c0s7 : Wed Aug 01 21:30:51 CDT 2012
   c0-0c1s1 : Wed Aug 01 21:30:51 CDT 2012
   .
   .
   .
   c0-0 : Wed Aug 01 21:30:52 CDT 2012
   ```

3. Verify that the `zone.tab` file in the `/usr/share/zoneinfo` directory contains the time zone you want to set.

   ```
   smw:~ # grep America/New_York /usr/share/zoneinfo/zone.tab
   US      +404251-0740023 America/New_York      Eastern Time
   ```

4. Create the time conversion information files.

```
smw:~ # date
Wed Aug 01 21:32:52 CDT 2012
smw:~ # /usr/sbin/zic -l America/New_York
smw:~ # date
Wed Aug 01 22:33:05 EDT 2012
```

5. Modify the `clock` file in the `/etc/sysconfig` directory to set the `DEFAULT_TIMEZONE` and the `TIMEZONE` variables to the new time zone.

```
smw:/etc/sysconfig # grep TIMEZONE /etc/sysconfig/clock
TIMEZONE="America/Chicago"
DEFAULT_TIMEZONE="US/Eastern"
smw:~ # vi /etc/sysconfig/clock
make changes
smw:~ # grep TIMEZONE /etc/sysconfig/clock
TIMEZONE="America/New_York"
DEFAULT_TIMEZONE="US/Eastern"
```

6. Copy the `/etc/localtime` file to `/opt/tfptboot`, and then restart the log system and `rsms`.

```
smw:~ # cp /etc/localtime /opt/tftpboot
smw:~ # /etc/init.d/cray-syslog restart
smw:~ # /etc/init.d/rsms restart
```

   For Cray XC30 systems, continue with step 7.

7. **For Cray XC30 systems only**: Reboot the cabinet controllers to get the updated time zone.

   a. Power down the system.

```
smw:~ # xtcli power down s0
```

   b. Reboot the cabinet controllers, then ensure that all cabinet controllers are up.

```
smw:~ # xtccreboot -c all
xtccreboot: reboot sent to specified CCs
smw:~ # xtalive -l cc
```

   c. Power up the system.

```
smw:~ # xtcli power up s0
```

      Note that at this point the `xtcli status` output shows that all nodes are "off", because they have not yet been bounced.

   d. Exit from the `root` login.

```
smw:~ # exit
```

8. Bounce the system.

```
crayadm@smw:~> xtbounce s0
```

Note: An incompatibility exists between the current version of `/etc/localtime` and earlier versions that may be on the system. This incompatibility causes the `date` command to report an incorrect time on the compute nodes. To resolve this incompatibility, after updating the SMW software you will also need to update the time zone on the compute nodes as described in the procedure *Changing the time zone for compute nodes* in *Installing and Configuring Cray Linux Environment (CLE) Software*.

Procedure 115. Changing the time zone on the boot root and shared root

Perform the following steps to change the time zone. You must be logged on as `root`. In this example, the time zone is changed from `"America/Chicago"` to `"America/New_York"`.

1. Confirm the time zone setting on the SMW.

```
smw:~ # cd /etc/sysconfig
smw:/etc/sysconfig # grep TIMEZONE clock
TIMEZONE="America/New_York"
DEFAULT_TIMEZONE="US/Eastern"
```

2. Log on to the boot node.

```
smw:/etc/sysconfig # ssh root@boot
boot:~ #
```

3. Verify that the `zone.tab` file in the `/user/share/zoneinfo` directory contains the time zone you want to set.

```
boot:~ # cd /usr/share/zoneinfo
boot:/usr/share/zoneinfo # grep America/New_York zone.tab
US      +404251-0740023 America/New_York        Eastern Time
```

4. Create the time conversion information files.

```
boot:/usr/share/zoneinfo # date
Mon Jul 30 22:50:52 CDT 2012
boot:/usr/share/zoneinfo # /usr/sbin/zic -l America\New_York
boot:/usr/share/zoneinfo # date
Mon Jul 30 23:59:38 EDT 2012
```

5. Modify the `clock` file in the `/etc/sysconfig` directory to set the `DEFAULT_TIMEZONE` and the `TIMEZONE` variables to the new time zone.

```
boot:/usr/share/zoneinfo # cd /etc/sysconfig
boot:~ # grep TIMEZONE clock
TIMEZONE="America/Chicago"
DEFAULT_TIMEZONE="US/Eastern"
boot:~ # vi clock
make changes
boot:~ # grep TIMEZONE clock
TIMEZONE="America/New_York"
DEFAULT_TIMEZONE="US/Eastern"
```

6. Switch to the default view by using `xtopview`.

   Note: If the SDB node has not been started, you must include the `-x /etc/opt/cray/sdb/node_classes` option when you invoke the `xtopview` command.

   ```
   boot:~ # xtopview
   default/:/ #
   ```

7. Verify that the `zone.tab` file in the `/user/share/zoneinfo` directory contains the time zone you want to set.

   ```
   default/:/ # grep America/New_York /usr/share/zoneinfo zone.tab
   US      +404251-0740023 America/New_York        Eastern Time
   ```

8. Create the time conversion information files.

   ```
   default/:/ # date
   Mon Jul 30 23:10:52 CDT 2012
   default/:/ # /usr/sbin/zic -l America/New_York
   default/:/ # date
   Tue Jul 31 00:11:38 EDT 2012
   ```

9. Modify the `clock` file in the `/etc/sysconfig` directory to set the `DEFAULT_TIMEZONE` and the `TIMEZONE` variables to the new time zone.

   ```
   default/:/ # cd /etc/sysconfig
   default/:/etc/sysconfig # grep TIMEZONE clock
   TIMEZONE="America/Chicago"
   DEFAULT_TIMEZONE="US/Eastern"
   default/:/etc/sysconfig # vi clock
   make changes
   default/:/etc/sysconfig # grep TIMEZONE clock
   TIMEZONE="America/New_York"
   DEFAULT_TIMEZONE="US/Eastern"
   ```

10. Exit `xtopview`.

    ```
    default/:/etc/sysconfig # exit
    boot:/usr/share/zoneinfo #
    ```

Procedure 116. Changing the time zone for compute nodes

1. Exit from the boot node and confirm the time zone setting on the SMW.

   ```
   boot:/usr/share/zoneinfo # exit
   smw:/etc/sysconfig # grep TIMEZONE clock
   TIMEZONE="America/New_York"
   DEFAULT_TIMEZONE="US/Eastern"
   ```

2. Copy the new `/etc/localtime` file from the SMW to the bootimage template directory.

   ```
   smw:/etc/sysconfig # cp -p /etc/localtime \
   /opt/xt-images/templates/default/etc/localtime
   ```

3. Copy the new /usr/share/zoneinfo file from the SMW to the bootimage template directory. The directory to contain the time zone file must be created in the bootimage template area.

```
smw:/etc/sysconfig # mkdir -p \
/opt/xt-images/templates/default/usr/share/zoneinfo/America
smw:~# cp -p /usr/share/zoneinfo/America/New_York \
/opt/xt-images/templates/default/usr/share/zoneinfo/America/New_York
```

   Note: This procedure enables a single time zone for the compute nodes. If users will be setting the TIMEZONE variable to time zones which are not the system default, you may wish to either copy a few of the common time zones used by the user community or the entire /usr/share/zoneinfo directory to the /opt/xt-images/templates/default/ area.

4. Update the boot image to include these changes; follow the steps in Procedure 2 on page 64.

   The time zone is not changed until you boot the compute nodes with the updated boot image.

Creating Modulefiles [E]

The Cray Programming Environment 2.0 release introduces the `craypkg-gen` utility, which makes use of the standard `pkg-config` utility, to configure the appropriate linking and compile flags and dependencies in `pkg-config` enabled modulefiles.

`craypkg-gen` assists with user and third party product integration with the compiler drivers (`cc`, `CC`, `ftn`) by creating `.pc` files for C, C++ and Fortran libraries and `pkg-config` enabled modulefiles.

Support for the `PRE_COMPILE_OPTS`, `PRE_LINK_OPTS`, `POST_COMPILE_OPTS`, `POST_LINK_OPTS`, and `INCLUDE_OPTS` environment variables is deprecated with the release of Cray Programming Environment 2.0, in favor of using `pkg-config` and `.pc` files to obtain this information from the library's package.

For more implementation details, see the `craypkg-gen`(1) man page. Please see http://www.freedesktop.org/wiki/Software/pkg-config for a `pkg-config` introduction.

PBS Professional Licensing for Cray Systems [F]

F.1 Introduction

PBS Professional uses a licensing scheme based on a central license server that allows licenses to float between servers. This reduces the complexity of managing environments with multiple, independent PBS installations and simplifies configuration when you run other software packages that use the same license manager.

The PBS server and scheduler run on the Cray service database (SDB) node. By default, the SDB node is only connected to the Cray system high-speed network (HSN) and cannot access an external license server. Various options to set up network connectivity between the license server and the SDB node are detailed below. Determine which option is best suited to your needs and implement that solution prior to installing the PBS Professional software from Altair.

Note: Regardless of the option chosen, you must run a PBS Professional MOM daemon on each login node where users may launch jobs.

PBS Professional configuration options on a Cray system include:

- **Running the PBS Professional server and scheduler on a Cray system service node.** If you choose to run the PBS Professional scheduler and server on a login node, you should be aware that these daemons consume processor and memory resources and have the potential to impact other processes running on the login node. In addition, any service running on a node where users are allowed to run processes increases the potential for interruption of that service. While these risks are generally low, it is important that you consider them before selecting this option. Refer to Migrating the PBS Professional Server and Scheduler on page 408 to configure PBS Professional using this strategy.
- **Moving the PBS Professional server and scheduler external to the Cray system.** The PBS Professional scheduler requests MPP data from one of the MOM daemons running on the Cray system login nodes. The volume of this data is dependent upon the size and utilization of the Cray system. If you run the PBS Professional scheduler outside of the Cray system, the scheduler cycle time could increase due to decreased bandwidth and increased latency in network communication. In most cases, the difference in cycle time is negligible.

However, if your system has larger node counts (> 8192), you may want to avoid this option. To configure PBS Professional for this strategy, refer to Migrating the PBS Professional Server and Scheduler on page 408.

- **Configuring the SDB node as an RSIP client.** This options allows you to leave the PBS Professional scheduler and server on the SDB node. If you are already running RSIP, this may be an attractive option. Cray recommends a dedicated network node for the RSIP server, which may not be desirable if you are not already running RSIP. Follow the appropriate procedure in Configuring RSIP to the SDB Node on page 410 to configure the SDB node as an RSIP client.
- **Configuring Network Address Translation (NAT) to forward IP packets to and from the SDB node.** This may be the best choice if you intend to use packet forwarding exclusively for PBS Professional licensing and do not mind running NAT services on a login node. The steps to configure NAT IP forwarding to the SDB node are described in Network Address Translation (NAT) IP Forwarding on page 413.
- **Installing a network card in the SDB node to connect it to the external network.** With this option you do not need to configure RSIP or NAT, but you must purchase a PCIe network interface card (NIC) for a modest cost. This is an attractive option if you want to access the SDB node directly from your external network. This procedure does not require connection through another node on the Cray system. The steps to configure this option are covered in Installing and Configuring a NIC on page 415.

Cray recommends that system administrators consult their local networking and security staff prior to selecting one of these options. Once you have chosen and configured a method for accessing the license server, complete the PBS Professional license manager configuration as described in the *Altair License Management System Installation Guide*. For additional information about using the `qmgr` command to set up the `pbs_license_file_location` resource, see the *PBS Professional Installation and Upgrade Guide* from Altair Engineering, Inc. For more information, see: http://www.pbsworks.com.

F.2 Migrating the PBS Professional Server and Scheduler

Before migrating the PBS Professional server and scheduler off of the SDB node you must first select the target host. PBS Professional versions 9.2 and beyond are MPP aware, meaning they are capable of scheduling jobs to Cray systems. If you already have a central PBS Professional server and scheduler, simply add the Cray system to the list of nodes.

The first step is to install PBS Professional on the Cray system as described in the *PBS Professional Installation and Upgrade Guide*. The install procedure configures the SDB node as the PBS Professional server and scheduler host. You must modify the default configuration to ensure that the PBS Professional scheduler and server do not start automatically on the SDB node.

Procedure 117. Migrating PBS off the SDB node

1. If the PBS scheduler and server are running on the SDB node, log on to the SDB and stop the services.

   ```
   sdb:~ # /etc/init.d/pbs stop
   ```

2. Log on to the Cray system boot node as root and unspecialize the PBS Professional configuration file for the SDB node. For example, your SDB is node 3, type the following commands:

   ```
   boot:~ # xtopview -m "Unspecialize pbs.conf on the SDB" -n 3
   node/3:/ # xtunspec /etc/pbs.conf
   node/3:/ # exit
   boot:~ #
   ```

3. Edit the PBS Professional configuration file for the login nodes to point to the new server. The new server may be one of the login nodes or a host external to the Cray system. Set PBS_SERVER in /etc/pbs.conf to the new PBS Professional server host. For example, if your server is named *myserver*, type the following commands:

   ```
   boot:~ # xtopview -m "Update pbs.conf for new server" -c login
   class/login/: # vi /etc/pbs.conf
   PBS_SERVER=myserver.domain.com
   class/login/: exit
   boot:~#
   ```

4. To migrate the server and scheduler to a login node and start PBS Professional automatically at boot time, specialize the /etc/pbs.conf file for that node. If the services are being moved to an external host, skip this step. For example, if the node ID of the login node is 4, type the following commands:

   ```
   boot:~ # xtopview -m "Specialize pbs.conf for new server" -n 4
   node/4:/ # xtspec /etc/pbs.conf
   ```

5. Modify the /etc/pbs.conf file to start all of the PBS Professional services; for example:

   ```
   node/4:/ # vi /etc/pbs.conf
   PBS_START_SERVER=1
   PBS_START_SCHED=1
   PBS_START_MOM=1

   node/4:/ # exit
   boot:~ #
   ```

6. Log on to each of the login nodes as root and modify the PBS Professional

MOM configuration file `/var/spool/PBS/mom_priv/config`. Change the `$clienthost` value to the name of the new PBS Professional server. For example, if your server is named *myserver*, type the following commands:

```
login2:~ # vi /var/spool/PBS/mom_priv/config
$clienthost myserver.domain.com
```

7. After the configuration file has been updated, restart PBS Professional on each login node.

   ```
   login2:~ # /etc/init.d/pbs restart
   ```

   Note: This command starts the PBS Professional scheduler and server if you have migrated them to a login node.

8. Log on to the new PBS Professional server host and add a host entry for each of the login nodes.

   ```
   myserver:~ # qmgr
   Qmgr: create node mycrayxt1
   Qmgr: set node mycrayxt1 resources_available.mpphost=xthostname
   Qmgr: create node mycrayxt2
   Qmgr: set node mycrayxt2 resources_available.mpphost=xthostname
   Qmgr: exit
   myserver:~
   ```

   At this point, the login nodes should be visible to the PBS Professional server.

F.3 Configuring RSIP to the SDB Node

Follow the instructions in this section to configure the SDB node as an RSIP client. Once the SDB node is configured as an RSIP client, refer to the *Altair License Management System Installation Guide* for detailed instructions about obtaining and installing the appropriate license manager components.

If you have not configured RSIP on your system, follow Procedure 118 on page 411 to generate a simple RSIP configuration with a single server and only the SDB node as a client.

Using the `CLEinstall` Program to Install and Configure RSIP on page 210 includes procedures to configure RSIP on a Cray system using the `CLEinstall` installation program. If you have already configured RSIP using these procedures during your Cray Linux Environment (CLE) installation or upgrade, follow Procedure 119 on page 412 to add the SDB node as an RSIP client for one of your existing RSIP servers.

For additional information on configuring RSIP services, see Configuring Realm-specific IP Addressing (RSIP) on page 209.

Procedure 118. Creating a simple RSIP configuration with the SDB node as a client

Important: Cray strongly recommends Procedure 48 on page 211 for RSIP configuration.

1. Boot the system as normal. Ensure all the service nodes are available, and ensure that the system is setup to allow password-less `ssh` access for the root user.
2. Select a service node to run the RSIP server. The RSIP server node must have external Ethernet connectivity and must not be a login node. In this example the physical ID for the RSIP server is `c0-0c0s6n1`.
3. Specialize the `rsipd.conf` file by node ID and install the `rsip.conf` file to the shared root. Additionally, tune the RSIP servers by updating the associated `sysctl.conf` file. Invoke the following steps for the RSIP server node.
   a. Log on to the boot node and invoke `xtopview` in the node view.

      ```
      boot:~ # xtopview -n c0-0c0s6n1
      node/c0-0c0s6n1:/ #
      ```

   b. Specialize `/etc/opt/cray/rsipd/rsipd.conf` for the specified node.

      ```
      node/c0-0c0s6n1:/ # xtspec /etc/opt/cray/rsipd/rsipd.conf
      ```

   c. Copy the `rsip.conf` template file from the SMW to the shared root.

```
node/c0-0c0s6n1:/ # scp crayadm@smw:/opt/cray-xt-rsipd/default/etc/rsipd.conf.example \
/etc/opt/cray/rsipd/rsipd.conf
```

   d. Modify the `port_range`, `ext_if` and `max_clients` parameters in the `rsipd.conf` file as follows:

      ```
      node/c0-0c0s6n1:/ # vi /etc/opt/cray/rsipd/rsipd.conf
      port_range 8192-60000
      max_clients 2
      Uncomment:
      ext_if eth0
      ```

      Note: If your external Ethernet interface is not `eth0`, modify `ext_if` accordingly. For example,

      ```
      ext_if eth1
      ```

   e. Specialize the `/etc/sysctl.conf` file and modify the OS port range so that it does not conflict with the RSIP server.

      ```
      node/c0-0c0s6n1:/ # xtspec /etc/sysctl.conf
      node/c0-0c0s6n1:/ # vi /etc/sysctl.conf
      net.ipv4.ip_local_port_range = 60001 61000
      ```

f. If the specified RSIP server is using a 10GbE interface, update the default socket buffer settings by modifying the following lines in the `sysctl.conf` file.

```
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.core.rmem_max = 524287
net.core.wmem_max = 524287
net.core.rmem_default = 131072
net.core.wmem_default = 131072
net.ipv4.tcp_rmem = 131072 1000880 9291456
net.ipv4.tcp_wmem = 131072 1000880 9291456
net.ipv4.tcp_mem = 131072 1000880 9291456
```

g. Update the `udev` rules to skip the `ifup` of the `rsip` interfaces as they are created. Add `rsip*` to the list of interface names for `GOTO="skip_ifup"`.

```
node/c0-0c0s6n1:/ # xtspec /etc/udev/rules.d/31-network.rules
node/c0-0c0s6n1:/ # vi /etc/udev/rules.d/31-network.rules
SUBSYSTEM=="net", ENV{INTERFACE}=="rsip*|ppp*|ippp*|isdn*|plip*|lo*|irda*| \
dummy*|ipsec*|tun*|tap*|bond*|vlan*|modem*|dsl*", GOTO="skip_ifup"
```

h. Exit the `xtopview` session.

```
node/c0-0c0s6n1:/ # exit
```

4. Update the boot automation script to start the RSIP client on the SDB node. This line is simply a `modprobe` of the `krsip` module with the IP argument pointing to the HSN IP address of the RSIP server node and specifying the requested number of ports; place the new line towards the end of the file, immediately before any `'motd'` or `'ip route add'` lines. For example, if the IP address of the RSIP server is `10.128.0.14` and 32 ports are requested, type the following commands.

```
crayadm@smw:~> vi /opt/cray/etc/auto.xthostname
# RSIP client startup
lappend actions { crms_exec_via_bootnode "sdb" "root" "modprobe krsip
ip=10.128.0.14use_rsip_local_ports=1 num_ports=32" }
```

Procedure 119. Adding the SDB node as an RSIP client to an existing RSIP configuration

1. Select one of your RSIP servers to provide RSIP access for the SDB node. In this example, we have chosen the RSIP server with the physical ID `c0-0c0s6n1`.
2. Log on to the boot node and invoke `xtopview` in the node view for the RSIP server you have selected.

```
boot:~ # xtopview -n c0-0c0s6n1
node/c0-0c0s6n1:/ #
```

3. Modify the `max_clients` parameters in the `rsipd.conf` file; Add 2 more clients to make room for the new SDB node. For example, if you configured 300 RSIP clients (compute nodes), type the following:

```
node/c0-0c0s6n1:/ # vi /etc/opt/cray/rsipd/rsipd.conf
max_clients 302
```

4. Update the boot automation script to start the RSIP client on the SDB node. Do this after the line that starts the RSIP server. This line is simply a `modprobe` of the `krsip` module with the IP argument pointing to the HSN IP address of the RSIP server node. For example, if the IP address of the RSIP server is `10.128.0.14`, type the following commands.

```
crayadm@smw:~> vi /opt/cray/etc/auto.xthostname
# RSIP client startup
lappend actions { crms_exec_via_bootnode "sdb" "root" "modprobe krsip
ip=10.128.0.14 rsip_local_ports=1" }
```

F.4 Network Address Translation (NAT) IP Forwarding

Follow Procedure 120 on page 413 to configure NAT IP forwarding for the SDB node.

Procedure 120. Configuring NAT IP forwarding for the SDB node

1. Select a login node to act as the NAT router. Cray recommends that you select the node with the lowest load or network latency. For this example the login node is named `login2`.

2. Log on to the node you have selected and invoke the `ifconfig` command to obtain the IP address of the routing node.

   If you have a Cray system with a Gemini-based, system interconnection network, type this command:

```
login2:/ # ifconfig ipogif0
ipogif0   Link encap:Ethernet  HWaddr 00:01:01:00:00:04
          inet addr:10.128.0.3  Mask:255.252.0.0
          inet6 addr: fe80::201:1ff:fe00:4/64 Scope:Link
          UP RUNNING NOARP  MTU:16000  Metric:1
          RX packets:1543290 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1640783 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1643894879 (1567.7 Mb)  TX bytes:1446996661 (1379.9 Mb)
```

   The IP address of the routing node is indicated as `inet addr` (in this case, *10.128.0.3*).

3. Record the Ethernet interface used on this login node. For example:

```
login2:/ # netstat -r  | grep default
default    cfgw-12-hsrp.us 0.0.0.0     UG      0 0        0 eth0
```

Following this example, use the Ethernet interface, `eth0`, in the NAT startup script that is created in the next step.

4. Edit `/etc/hosts` on the shared root to include the external license server(s). Add these entries prior to the first local Cray IP addresses; that is, before the `10.128.`*x.y* entries. For example:

```
boot:~# xtopview
default/:/ # vi /etc/hosts
10.0.0.55    tic tic.domain.com
10.0.0.56    tac tac.domain.com
10.0.0.57    toe toe.domain.com
default/:/ # exit
boot:~#
```

5. In the same manner, edit `/etc/hosts` on the boot root to include entries for the external license server(s).

```
boot:~# vi /etc/hosts
10.0.0.55    tic tic.domain.com
10.0.0.56    tac tac.domain.com
10.0.0.57    toe toe.domain.com
```

6. In the default `xtopview` view, create and/or edit the `/etc/init.d/local.start-nat` file on the shared root, adding the following text:

```
boot:~# xtopview
default/:/ # vi /etc/init.d/local.start-nat

#!/bin/bash
### BEGIN INIT INFO
# Provides:       local.start-nat
# Required-Start:
# Required-Stop:
# Default-Start:
# Default-Stop:
# Description:    Set up NAT IP forwarding
### END INIT INFO

echo "Setting up NAT IP forwarding."
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -F
iptables -A FORWARD -i eth0 -o ipogif0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i ipogif0 -o eth0 -j ACCEPT
iptables -A FORWARD -j LOG
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

7. Add execute permissions to the `local.start-nat` file:

```
default/:/ # chmod 755 /etc/init.d/local.start-nat
```

8. Exit the `xtopview` session.

```
default/:/ # exit
boot:~#
```

9. Log on as `root` to the selected router node and start the NAT service. Use the `iptables` command to verify that forwarding is active.

```
login2:~ # /etc/init.d/local.start-nat
login2:~ # iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
LOG        all  --  0.0.0.0/0            0.0.0.0/0           LOG flags 0 level 4

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
login2:~ #
```

10. Add a new default route on the SDB node. Ensure that this route does not currently exist. For example, if the routing node IP interface you identified in step 2 is *10.128.0.3*, type this command:

    ```
    login2:~ # ssh sdb /sbin/route add default gw 10.128.0.3
    ```

11. Test the new route by invoking the `ping` command and ensuring the service node can access external servers by name.

12. Edit the boot automation script to Configure NAT services. For example, if the IP address you identified in step 2 is *10.128.0.3*:

    ```
    smw:~# vi /opt/cray/etc/auto.xthostname
    ```

    Add the following lines just prior to the ALPS/PBS startup:

    ```
    lappend actions { crms_exec_via_bootnode "login2" "root" \
    "/etc/init.d/local.start-nat" }
    lappend actions { crms_exec_via_bootnode "sdb" "root" \
    "/sbin/route add default gw 10.128.0.3" }
    ```

    NAT services should now restart automatically upon the next reboot of the Cray system.

F.5 Installing and Configuring a NIC

Obtain a PCIe compliant NIC. Intel 82546 based cards have been verified with Cray system SDB nodes. Follow Procedure 121 on page 415 to install the network card in the SDB node and connect it to the external network. Note that you are required to reboot your system as part of this procedure.

Procedure 121. Installing and configuring a NIC on the SDB node

1. Prior to shutting the system down, perform the following steps on the boot node to ensure the new NIC is configured upon the ensuing reboot. Invoke `xtopview` in the node view for the SDB node. For example, if your SDB is node *3*, the

IP address to assign on the external network is *172.30.10.100*, the appropriate `netmask` is *255.255.0.0*, and the default gateway IP is *172.30.10.1*, type these commands.

```
boot:~# xtopview -m "add eth0 interface" -n 3
node/3:/ # cd /etc/sysconfig/network
node/3:/ # xtspec ifcfg-eth0
node/3:/ # vi ifcfg-eth0
```

Add the following content to the ifcfg-eth0 file:

```
DEVICE="eth0"
BOOTPROTO="static"
STARTMODE="onboot"
IPADDR=172.30.10.100
NETMASK=255.255.0.0
node/3:/ # touch routes
node/3:/ # xtspec routes
node/3:/ # echo 'default 172.30.10.1 - -' >routes
node/3:/ # exit
boot:~ #
```

2. Edit the `/etc/hosts` file on the shared root and add entries for the external license server(s). For example:

```
boot:~# xtopview
default/:/ # vi /etc/hosts
```

Add these entries prior to the first local Cray system IP addresses; that is, before the 192.168.x.y entries.

```
10.0.0.55    tic tic.domain.com
10.0.0.56    tac tac.domain.com
10.0.0.57    toe toe.domain.com

default/:/ # exit
boot:~# exit
```

3. Shut down the system.

```
smw:~# xtbootsys -s last -a auto_xtshutdown
```

4. Power down the slot where the SDB node is installed. For example:

```
smw:~# xtcli power down_slot -f c0-0c0s0
```

5. Pull the blade, physically insert the new NIC into the PCIe slot of the SDB node and reinsert the blade into the slot.

6. Power up the slot where the SDB node is installed. For example:

```
smw:~# xtcli power up_slot -f c0-0c0s0
```

7. Connect the NIC to the Ethernet network on which the license server is accessible.

8. Boot the Cray system.

9. Log on to the SDB node and invoke the `ifconfig` command to confirm that the SDB shows the new `eth0` interface.

```
nid00003:~ #  /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:04:23:DF:4C:56
          inet addr:172.30.10.100  Bcast:172.30.10.1 Mask:255.255.0.0
          inet6 addr: 2001:408:4000:40c:204:23ff:fedf:4c56/64 Scope:Global
          inet6 addr: 2600:805:100:40c:204:23ff:fedf:4c56/64 Scope:Global
          inet6 addr: fe80::204:23ff:fedf:4c56/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:428807 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10400 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:34426088 (32.8 Mb)  TX bytes:1292332 (1.2 Mb)
          Base address:0x2fc0 Memory:fece0000-fed00000
```

10. Confirm that you can ping the license server from the SDB node.

```
nid00003:~ #  ping tic.domain.com
```

Installing RPMs [G]

A variety of software packages are distributed as standard Linux RPM Package Manager (RPM) packages. RPM packages are self-contained installation files that must be executed with the `rpm` command in order to create all required directories and install all component files in the correct locations.

G.1 Generic RPM Usage

To install RPMs on a Cray system, you must use `xtopview` on the boot node to access and modify the shared root. The `rpm` command is not able to modify the RPM database from a login node or other service node; the root directory is read-only from these nodes.

Any changes to the shared root apply to all service nodes. If the RPM you are installing modifies files in `/etc`, you must invoke `xtopview` to perform any class or node specialization that may be required. `xtopview` specialization applies only to `/etc` in the shared root.

For some Cray distributed RPMs, you can set the `CRAY_INSTALL_DEFAULT` environment variable to configure the new version as the default. Set this variable before you install the RPM. For more information, see the associated installation guide.

For more information on installing RPMs, see the `xtopview`(8) man page and the installation documentation for the specific software package you are installing.

Example 132. Installing an RPM on the SMW

As `root`, use the following command:

```
smw:~# rpm -ivh /directorypath/filename.rpm
```

Example 133. Installing an RPM on the boot node root

As `root`, use the following command:

```
boot:~ # rpm -ivh /directorypath/filename.rpm
```

Example 134. Installing an RPM on the shared root

As `root`, use the following commands:

Note: If the SDB node has not been started, you must include the `-x /etc/opt/cray/sdb/node_classes` option when you invoke the `xtopview` command.

```
boot:~ # cp -a /tmp/filename.rpm /rr/current/software
boot:~ # xtopview
default/:/ # rpm -ivh /software/filename.rpm
```

Sample LNET Router Controller Script [H]

```
#Lnet.rc
#!/bin/bash
#
# $Id: lnet.rc bogl Exp $
#
### BEGIN INIT INFO
# Provides:          lnet
# Required-Start:    $network openibd
# Required-Stop:     $network openibd
# X-UnitedLinux-Should-Start:
# Default-Start:     3
# Default-Stop:      0 1 2 5 6
# Description:       Enable lnet routers
### END INIT INFO
#set -x
PATH=/bin:/usr/bin:/usr/sbin:/sbin:/opt/cray/lustre-cray_ari_s/default/sbin
. /etc/rc.status
rc_reset
case "$1" in
    start)
         echo -n "Starting lnet "
         modprobe lnet
         lctl net up > /dev/null
         rc_status -v
         ;;
    stop)
         echo -n "Stopping lnet "
         lctl net down > /dev/null
         lustre_rmmod || true
         rc_status -v
         ;;
    restart)
         $0 stop
         $0 start
         rc_status
         ;;
    *)
         echo "Usage: $0 {start|stop|restart}"
         exit 1
         ;;
esac
rc_exit
```

Enable an Integrated Dell™ Remote Access Controller (iDRAC6) on a Rack-mount SMW [I]

Enabling an Integrated Dell Remote Access Controller (iDRAC6) allows you to manage your rack-mount SMW remotely.

I.1 Before You Start

Before you enable an iDRAC6 on a rack-mount SMW, you must:

- Have physical access to the SMW console
- Know your iDRAC6 IP address, subnet mask, and default gateway
- Know your SMW `root` account password

I.2 Enable an Integrated Dell Remote Access Controller (iDRAC6) on a Rack-mount SMW

Procedure 122. Change the BIOS and iDRAC settings

Use the following procedure to change the BIOS and iDRAC settings.

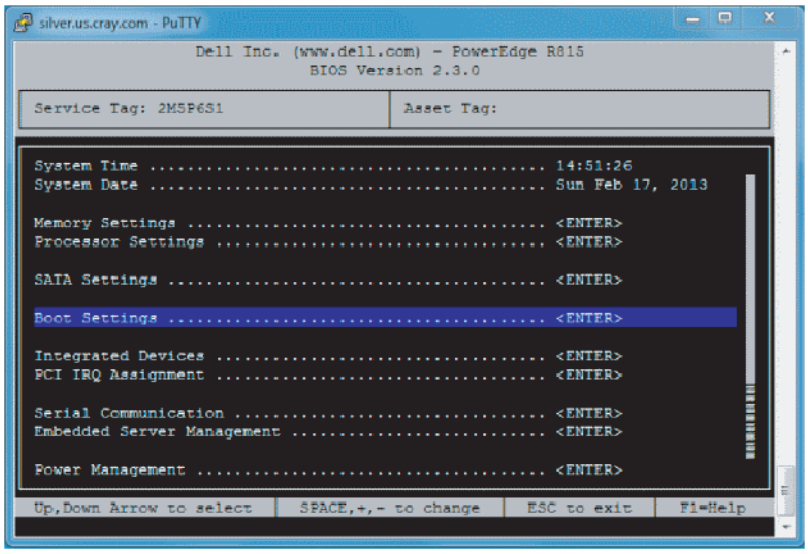
1. Power up the idrac. When the BIOS power-on self-test (POST) process begins, **quickly press the F2 key** after the following messages appear in the upper-right of the screen.

   ```
        F2 = System Setup
      F10 = System Services
   F11 = BIOS Boot Manager
           F12 = PXE Boot
   ```

   When the `F2` keypress is recognized, the `F2 = System Setup` line changes to `Entering System Setup`.

2. Select **Boot Settings**, then press `Enter`.

Figure 12. Rack-mount SMW Boot Settings Menu



a. Select **Boot Sequence**, then press `Enter` to view the boot settings.

b. In the pop-up window, change the boot order so that the integrated NIC appears first, before the optical (DVD) drive. The hard drive should be last on the list.

Figure 13. Rack-mount SMW Boot Sequence Menu

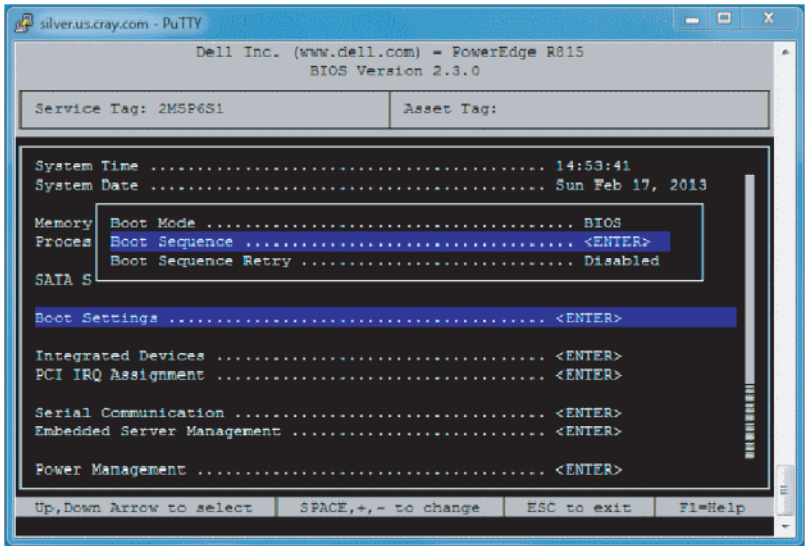
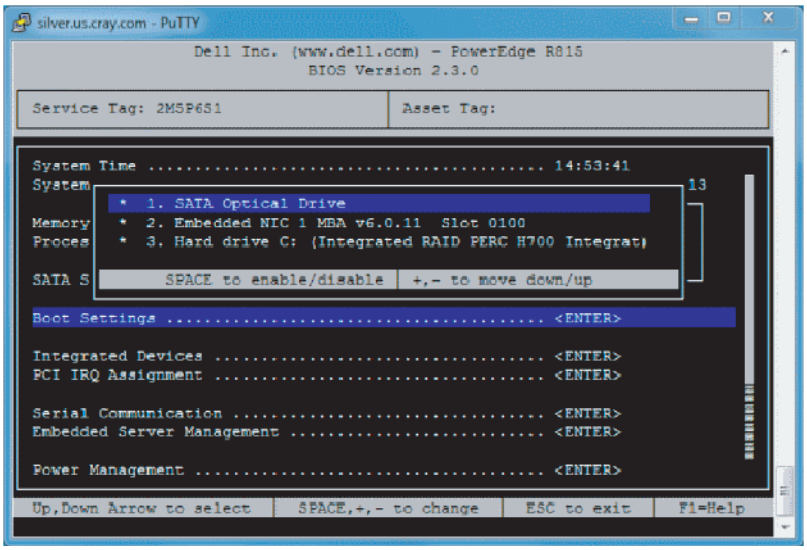


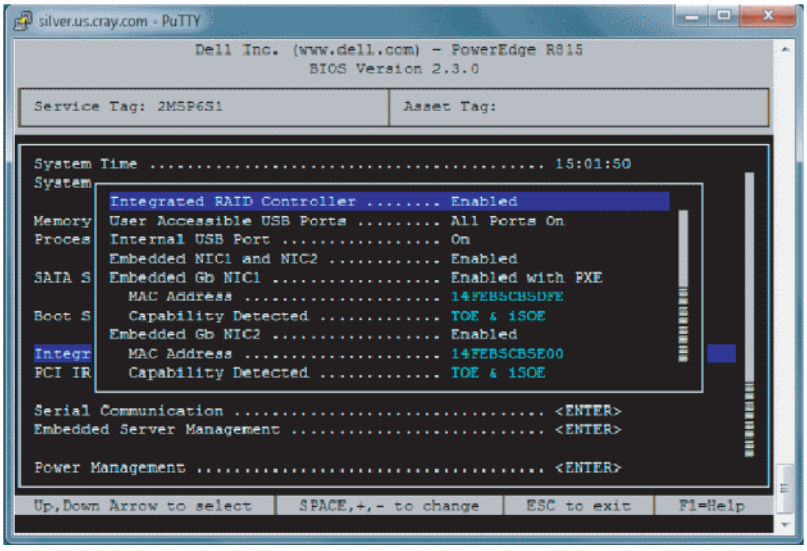
Figure 14. Rack-mount SMW Boot Sequence Settings



c. Press `Enter` to return to the **BIOS Boot Settings** screen.

3. Press `Esc` to return to the System Setup Menu, scroll down and select **Integrated Devices**.

Figure 15. Rack-mount SMW Integrated Devices (NIC) Settings



a. Set **Embedded Gb NIC 2** to **Enabled**.

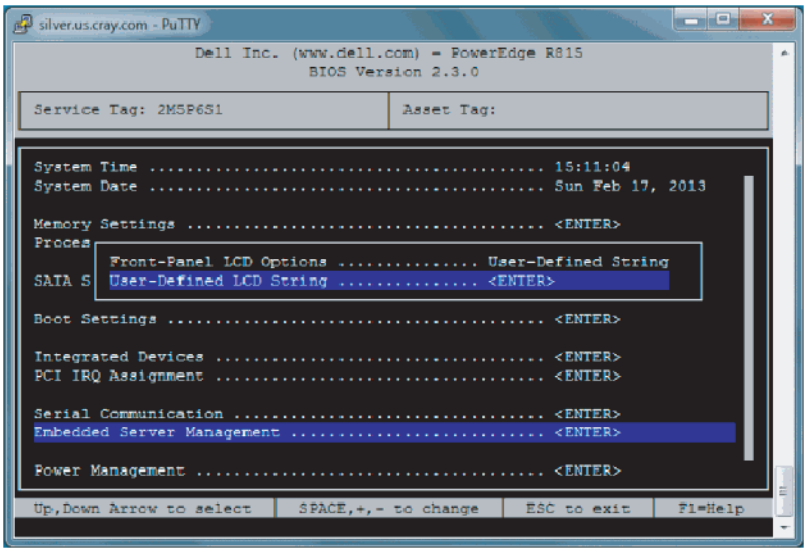
b. Scroll down and set **Embedded NIC 3** to **Enabled**.

c. Set **Embedded Gb NIC 4** to **Enabled**.

d. Press `Esc` to return to the System Settings Menu.

4. Select **Embedded Server Management**.

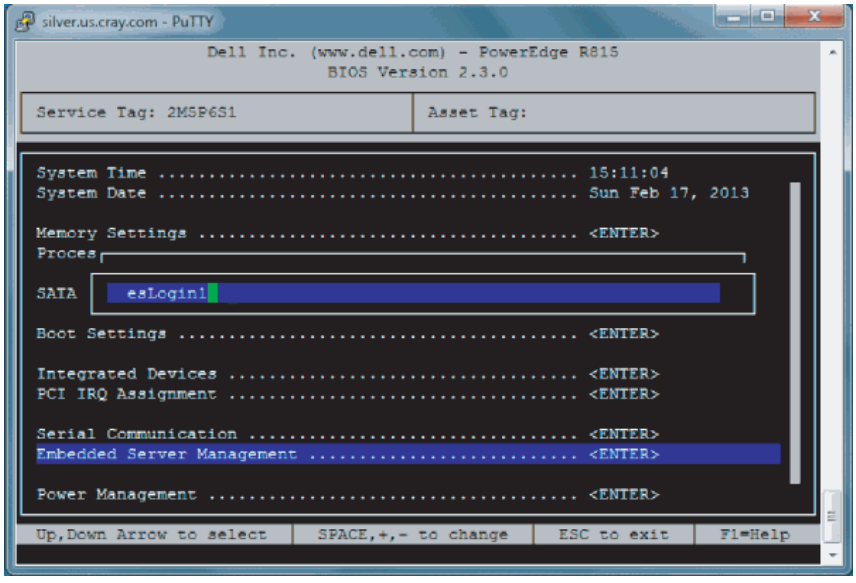
Figure 16. Rack-mount SMW Embedded Server Management Settings



a. Set **Front-Panel LCD Options** to **User-Defined LCD String**.

b. Set **User-Defined LCD String** to your login host name, such as `smw-drac`.

Figure 17. Rack-mount SMW User-defined LCD String Settings



5. Save your changes and exit.

a. Press `Escape` to exit the System Setup Main Menu.

b. The utility displays the prompt "Are you sure you want to exit and reboot?" Select **Yes**.

6. When the system reboots, press `Ctrl-E` to configure the iDRAC port settings.

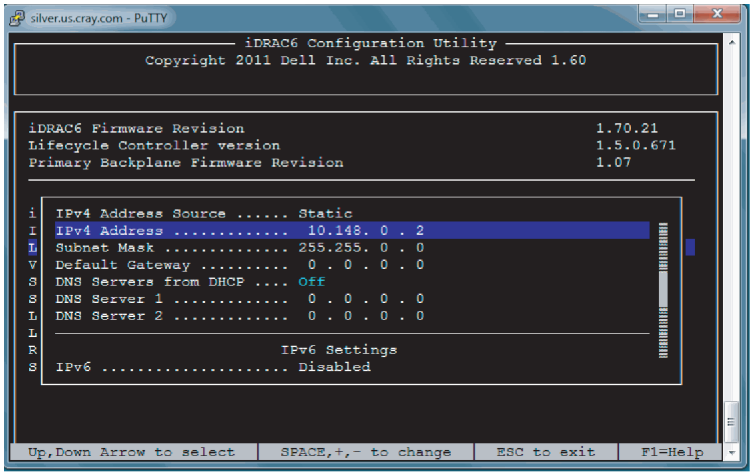
```
www.dell.com

iDRAC6 Configuration Utility 1.60
Copyright 2011 Dell Inc. All Rights Reserved
Four 2.10 GHz Twelve-core Processors, L2/L3 Cache: 6 MB/10 MB
iDRAC6 FirmwareaRevisionHversion: 1.70.21
.
.
.'
 IPv4 Stack      : Enabled
 IP Address      :  10.148. 0 . 2
 Subnet mask     : 255.255. 0 . 0
 Default Gateway :  0 . 0 . 0 . 0
Press <Ctrl-E> for Remote Access Setup within 5 sec......
```

a. Set the **iDRAC6 LAN** to **ON**.

b. Select **LAN Parameters** and press `Enter`. Set the IPv4 address to the SMW DRAC IP address.

c. Press `Esc` to return to the iDRAC 6 menu, and `Esc` to exit and save.

Figure 18. Rack-mount SMW DRAC IPv4 Parameter Settings



Procedure 123. Enabling an Integrated Dell Remote Access Controller (iDRAC6) on a rack-mount SMW

1. If the SMW is up, `su` to `root` and shut it down.

```
crayadm@smw:~> su - root
smw:~ # shutdown -h now;exit
```

2. Connect Ethernet cable to the iDRAC6 port. The cable is located on back of a rack-mount SMW in the lower left corner.
3. Power up the SMW.
4. After the BIOS, Dell PowerEdge Expandable RAID Controller (PERC) card, and disk map have displayed, the IPv4/IPv6 information displays. When the IPv4/IPv6 information displays, press `Ctrl-E`.
5. Using the arrow keys, select **LAN Parameters**, then press `Enter`.
6. Select **NIC Selection** and set it to **Dedicated**. Then press `Esc`.
7. Using the arrow keys, scroll down and select the **IPv4 settings** section.
   a. Ensure that IPv4 is enabled.
   b. Confirm that the IPv4 address source is set to static:

      ```
      IPv4 Address Source: Static
      ```

   c. Enter your iDRAC6 IP addresses for the following:
      - Address:
      - Subnet Mask:
      - Default Gateway:

   d. Ensure that IPv6 is disabled.
   e. Press `Esc` and return to the **LAN Parameters** window.
8. Using the arrow keys, select **LAN User Configuration**, then press `Enter`.

   Note: This configuration is for both SSH and web browser access to the iDRAC.

   a. Enter the `root` account name and iDRAC password.

      ```
      Account User name: root
      Enter Password: ******
      Confirm Password: ******
      ```

   b. Press `Esc`.
9. Press `Esc` again.
10. Select **Save Changes and Exit**, then press `Enter`. The SMW will complete booting up; no user interaction is required.

Procedure 124. Changing the default iDRAC Password

1. Log into the web interface as `root`.
2. Select **iDRAC settings** on the left-hand bar.
3. Select **network/Security** on the main top bar.

4. Select **Users** on the secondary top bar.
5. Select the user whose password you are changing. For example, userid `2` and username `root`.
6. Select **Configure User**, then **Next**.
7. Type the new password into the **New Password** and **Confirm New Password** text boxes.
8. Select **Apply** to complete the password change.

I.3 Using the iDRAC6

Procedure 125. Using the iDRAC6

1. Bring up a web browser.
2. Go to: `https://`*cray-drac*. A login screen appears.
3. Enter the account user name and password that you set up in Procedure 123 on page 427, step 8.a. Then Select **Submit**.

   The **System Summary** window appears.
4. To access the SMW console, select the **Console Media** tab.

   The **Virtual Console and Virtual Media** window appears.
5. Select **Launch Virtual Console**.

Tip: By default, your console window has two cursors: one for the console and one for your own window environment. To switch to single-cursor mode, select **Tools**, then **Single Cursor**. This single cursor will not move outside the console window. To exit single-cursor mode, press the `F9` key.

Tip: To logout of the virtual console, kill the window or select **File**, then **Exit**. You will still be logged into the iDRAC6 in your web browser.

For detailed information, download the iDRAC6 documentation at:

http://goo.gl/4Jm4T

Rack-mount SMW: Replace a Failed LOGDISK or DBDISK Disk Drive [J]

J.1 Rack-mount SMW: Replace a Failed LOGDISK or DBDISK Disk Drive

Warning: You **must** be running the SUSE Linux Enterprise Server version 11 Service Pack 3 (SLES 11 SP3) SMW base operating system and a release of SMW 7.2 or later on your SMW in order to perform the procedures in this chapter.

Procedure 126. Rack-mount SMW: Replace a failed LOGDISK or DBDISK disk drive

Note: This procedure specifies replacing the LOGDISK disk. If you are replacing the DBDISK, use the appropriate /dev/disk/by-path/pci* device name.

1. Replace the failed drive with the new drive.
2. Reboot the SMW.

   ```
   smw:~ # reboot
   ```

3. Reconfigure LOGDISK.

```
smw:~ # /sbin/fdisk /dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0
Command (m for help): p
```

   a. Delete all the current partitions, if there are any.

   ```
   Command (m for help): d
   Partition number 4
   Command (m for help): d
   Partition number 3
   Command (m for help): d
   Partition number 2
   Command (m for help): d
   Partition number 1
   Command (m for help): p
   ```

b. Create the new, single partition.

```
Command (m for help): n
Command action
 e   extended
 p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-121601, default 1): # Hit return, take the default
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-121601, default 121601):
# Hitreturn, take the default
Using default value 121601

Command (m for help): p

Disk /dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0:1000.2 GB, \
1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00000083

Device Boot                                                     Start   End    Blocks  Id System
/dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0-part1 1 121601 976760001 83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

4. Recreate the file system.

```
smw:~ # mkfs -t ext3 -b 4096
/dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000-lun-0-part1
```

5. Mount the newly created file system.

```
smw:~ # mount /dev/disk/by-path/pci-0000:05:00.0-sas-phy4-0x4433221104000000\
-lun-0-part1/ var/opt/cray/disk/1
```

6. The symbolic links should already be there to link this to `/var/opt/cray/log`.

   ```
   smw:~ # ls -al /var/opt/cray
   ```

7. Create the following new directories:

   ```
   smw:~> mkdir /var/opt/cray/disk/1/log
   smw:~> mkdir /var/opt/cray/disk/1/debug
   smw:~> mkdir /var/opt/cray/disk/1/dump
   ```

8. Restart the `rsms` daemon.

   ```
   smw:~ # /etc/init.d/rsms restart
   smw:~ # /etc/init.d/dbMonitor restart
   ```