XC™ Series Boot Troubleshooting Guide (CLE 6.0.UP03) S-2565

Contents

About the XC™ Series Boot Troubleshooting Guide

Scope and Audience

The *XC™ Series Boot Troubleshooting Guide* (S-2565) provides guidance and instructions to aid in troubleshooting a failed boot of an XC Series system running software releases CLE 6.0 and SMW 8.0 or later.

This publication is intended for system installers, administrators, and anyone who installs, configures, or manages software on a Cray XC™ Series system. It assumes some familiarity with standard Linux and open source tools (e.g., zypper/yum for RPMs, Ansible, YAML/JSON configuration data).

CLE 6.0.UP02 / CLE 6.0.UP02 / SMW 8.0.UP02 Release

XC™ Series Boot Troubleshooting Guide (CLE 6.0.UP02) S-2565 supports Cray software release CLE 6.0.UP02 / CLE 6.0.UP02 / SMW 8.0.UP02 for Cray XC™ Series systems, released on 08 012 2016.

New in this release

- New topic: *About Boot Automation Files* on page 43
- New topic: *About Cray Scalable Services* on page 25
- New content describing the capability to boot all service and compute nodes at the same time (after boot and SDB nodes are booted) in this topic: *Anatomy of an XC System Boot with xtbootsys* on page 27

Command Prompt Conventions

Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt.

- The `root` or super-user account always has the `#` character at the end of the prompt.
- Any non-`root` account is indicated with *`account@hostname>`*. A user account that is neither `root` nor `crayadm` is referred to as `user`.

`smw#`	Run the command on the SMW as `root`.
`cmc#`	Run the command on the CMC as `root`.
`sdb#`	Run the command on the SDB node as `root`.
`crayadm@boot>`	Run the command on the boot node as the `crayadm` user.
`user@login>`	Run the command on any login node as any non-`root` user.

`hostname#`	Run the command on the specified system as `root`.
`user@hostname>`	Run the command on the specified system as any non-`root` user.
`smw1#` `smw2#`	For a system configured with the SMW failover feature there are two SMWs—one in an active role and the other in a passive role. The SMW that is active at the start of a procedure is *smw1*. The SMW that is passive is *smw2*.
`smwactive#` `smwpassive#`	In some scenarios, the active SMW is *smw1* at the start of a procedure—then the procedure requires a failover to the other SMW. In this case, the documentation will continue to refer to the formerly active SMW as *smw1*, even though *smw2* is now the active SMW. If further clarification is needed in a procedure, the active SMW will be called *smwactive* and the passive SMW will be called *smwpassive*.

Command prompt inside chroot

If the `chroot` command is used, the prompt changes to indicate that it is inside a chroot environment on the system.

```
smw# chroot /path/to/chroot
chroot-smw#
```

Directory path in command prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the `cd` command is used to change into the directory, and the directory is referenced with a period (`.`) to indicate the current directory.

For example, here are actual prompts as they appear on the system:

```
smw:~ # cd /etc
smw:/etc# cd /var/tmp
smw:/var/tmp# ls ./file
smw:/var/tmp# su - crayadm
crayadm@smw:~> cd /usr/bin
crayadm@smw:/usr/bin> ./command
```

And here are the same prompts as they appear in this publication:

```
smw# cd /etc
smw# cd /var/tmp
smw# ls ./file
smw# su - crayadm
crayadm@smw> cd /usr/bin
crayadm@smw> ./command
```

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.

`Monospaced Bold`	Indicates commands that must be entered on a command line or in response to an interactive prompt.
`Oblique` or *`Italics`*	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a graphical user interface window or element and key strokes (e.g., **Enter**, **Alt-Ctrl-F**).
\ (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line). Do not type anything after the backslash or the continuation feature will not work correctly.

Feedback

Your feedback is important to us. Visit the Cray Publications Portal at *http://pubs.cray.com* and make comments online using the **Contact Us** button in the upper-right corner, or email comments to pubs@cray.com.

Trademarks

Introduction to Troubleshooting a Boot of an XC™ Series System

To troubleshoot failures encountered while booting an XC system, it is important to understand how system components are logically connected, what tasks are normally done during a successful boot, the interactions among daemons running on different parts of the XC system, and where to find log files to analyze a boot session.

Because the software, the steps within the booting process, and the interactions among daemons may change over time, this guide attempts to stress troubleshooting techniques rather than list specific error messages or conditions with any temporary workarounds that may later be integrated into software fixes.

Although this guide is intended to help primarily with boot difficulties, the topics on Ansible and config set troubleshooting may be useful outside the context of a boot as well, if difficulties are encountered while reconfiguring a system without booting it.

- *SMW and CLE Hardware Configuration and Cabling Concepts* on page 9: Describes how the SMW and CLE components are connected to each other for systems with a stand-alone SMW, SMW HA, and eLogin.
- *SMW Daemons, Processes, and Logs* on page 13:
  - Lists all daemons and processes that run on the SMW and are involved in the boot process.
  - Lists all logs that contain boot-related information and where to find them. Important information is logged during a boot, which helps with both troubleshooting and submitting a bug to Cray.
  - Describes how time is synchronized among XC system components.
- *Anatomy of an XC System Boot with xtbootsys* on page 27: Provides a step-by-step guide to the tasks performed during a typical boot.
- *The Booting Process from the CLE Node View* on page 45:
  - Describes three ways to boot CLE nodes: PXE boot (boot and SDB nodes only), tmpfs boot (all other service nodes, and an option for compute nodes), and Netroot boot (option for compute and login nodes only).
  - Lists all cray-ansible and Ansible logs and where to find them.
- *Commands Helpful in Troubleshooting a Boot* on page 53: Lists commands that help in investigating what went wrong. Use this as a reference.
- *Techniques for Troubleshooting a Failed Boot* on page 69: Lists problems that might be encountered and what to investigate to find the root cause. References many of the "helpful commands" from the previous section.
- *Information to Gather for Opening a Bug* on page 99: Describes what information is most helpful to include and how to obtain it.

SMW and CLE Hardware Configuration and Cabling Concepts

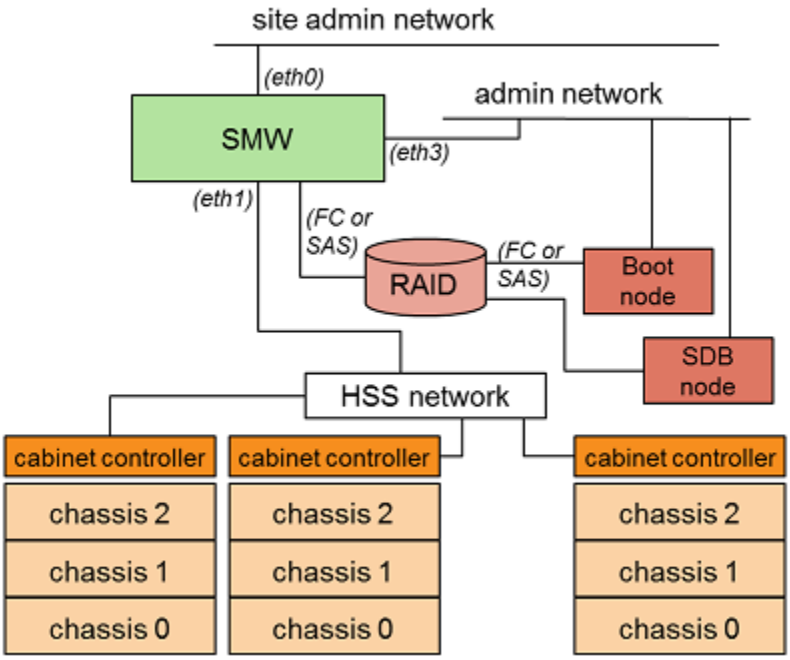
A familiarity with the hardware and cabling of an XC™ Series system may be helpful when troubleshooting a boot. These sections describe the basic hardware components and configurations for a system with a stand-alone SMW, an SMW HA (high availability) system, and a system that includes eLogin capability. The final section is a *A Guide to Hardware Component Names*.

Stand-alone SMW

This diagram shows a simplified view of the principal System Management Workstation (SMW) and Cray Linux Environment (CLE) hardware components for an XC™ Series system with a single, stand-alone SMW. The site admin or management network (eth0) connects to the system through the SMW only. The SMW, boot node, and SDB (service database) node are connected to each other over the admin network (eth3) and are each typically connected to the boot RAID with a Fibre Channel (FC) or serial attached SCSI (SAS) switch.

The SMW has a single Ethernet interface that talks to the cabinet controllers (CC) over an Ethernet switch. Each cabinet is on its own subnet, and routes are set up on the SMW to go over eth1 to get to each subnet. The SMW eth1 interface has the address 10.1.1.1/16 on the Hardware Supervisory System (HSS) network.

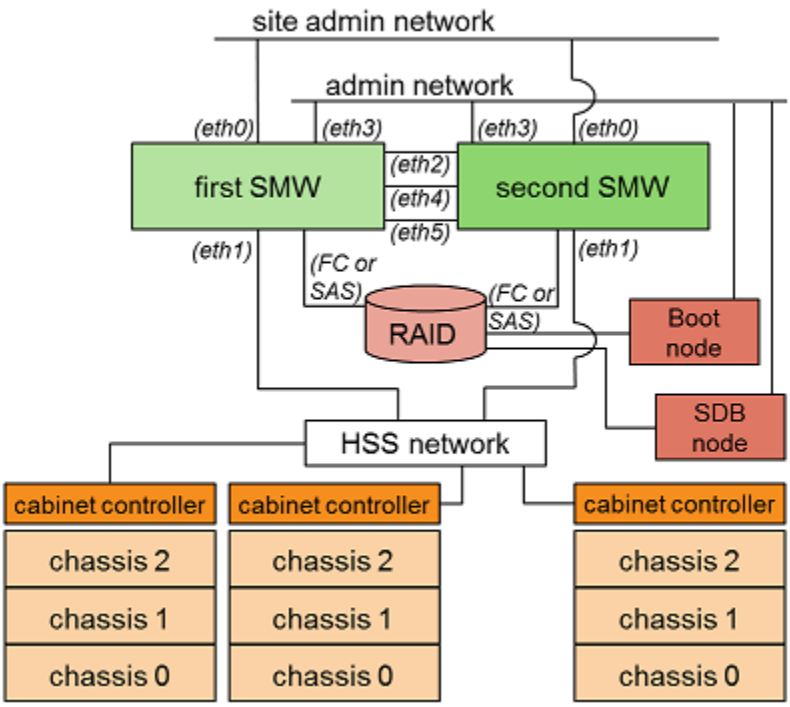
Figure 1. SMW and CLE Hardware Configuration



SMW HA

This diagram shows a simplified view of the principal components of an SMW HA system, which features two SMWs using cluster management software (SuSE High Availability Extension) in active/passive mode. A second SMW can assume the duties of the first SMW in the event of a software or hardware fault on the first SMW. Heartbeat networks are eth2 and eth4, while a distributed replicated block device (DRBD) is on an eth5 network.

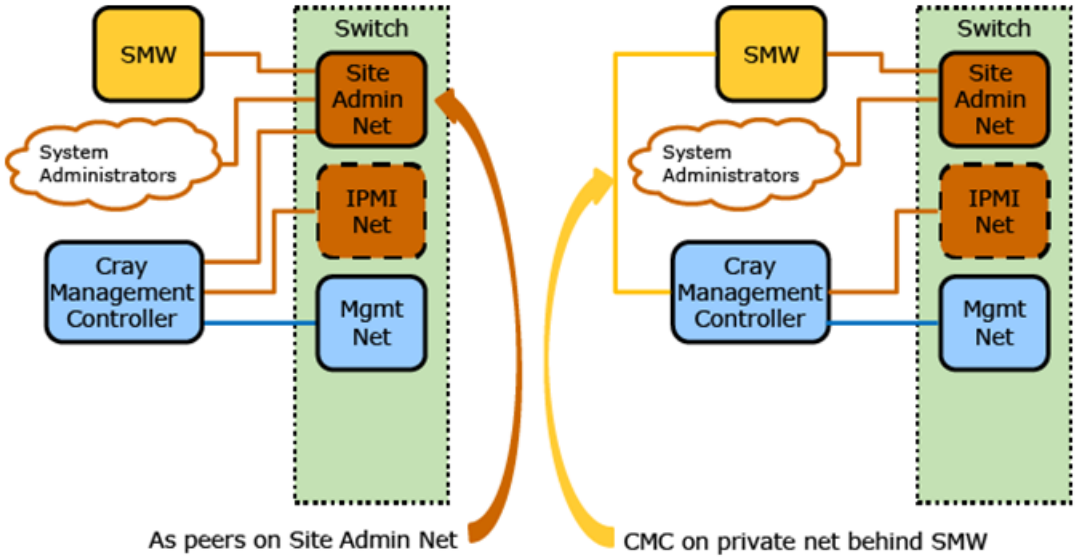
Figure 2. SMW and CLE Hardware Configuration with High Availability



eLogin

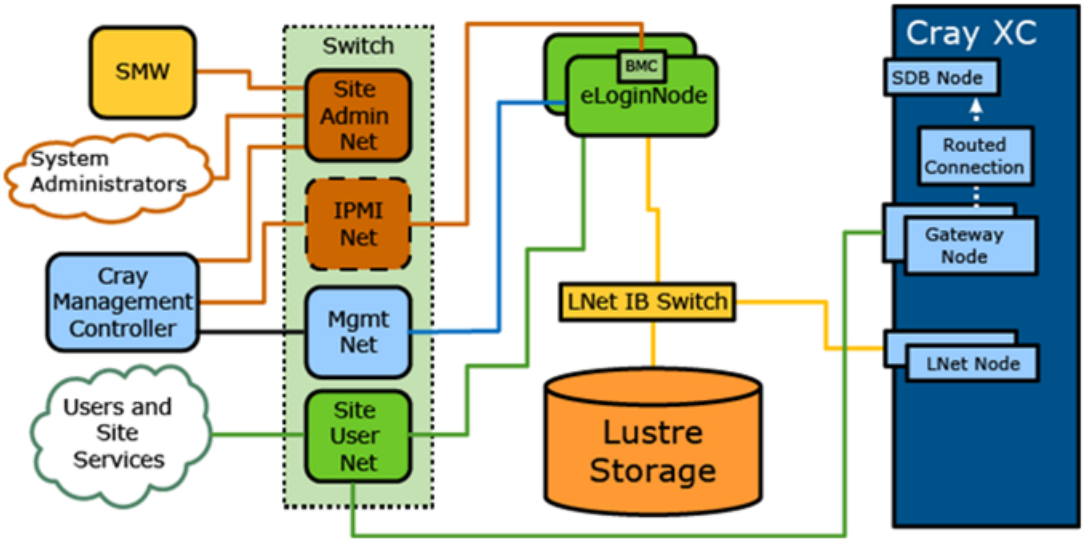
Sites that use eLogin have several options for configuring their systems. This first diagram shows two ways to connect the SMW to the Cray Management Controller (CMC).

Figure 3. Connecting SMW to CMC for eLogin



This diagram shows the cabling of Ethernet, FC/SAS to boot RAID, Infiniband (IB) to external Lustre, and the high speed network (HSN). Here, the eLogin nodes are on the site user network and access the SDB using a routed connection through the gateway nodes.

Figure 4. eLogin Nodes with Routed SDB and User Access through Gateway Nodes



For other common configurations (e.g., eLogin nodes connected directly to the SDB node) and a description of the networks used, see *XC™ Series eLogin Installation Guide* (S-2566) at *http://pubs.cray.com*.

A Guide to Hardware Component Names

The HSS, which is controlled by the state manager, provides a naming hierarchy for hardware components. These component names for the hardware are sometimes referred to as *cnames* (note that in the global or CLE

config set, *cname* usually means a single node). In the Format column of this table, capital letters represent numbers, such as in c1-2c0s4n3.

Table 1. Naming Hierarchy of XC™ Series Hardware Components

Component	Format	Description
System	s0, p0	All components attached to the SMW.
Cabinet	c*X*-*Y*	Cabinet column (*X*) and row (*Y*). This is the cabinet controller (CC) hostname.
Chassis	c*X*-*Y*c*C*	Physical chassis in cabinet, where *C* = 0, 1, or 2.
Blade or slot	c*X*-*Y*c*C*s*S*	Physical blade slot in chassis, where *S* = 0–15. This is the blade controller (BC) hostname.
Node	c*X*-*Y*c*C*s*S*n*N*	Node on a blade, where *N* = 0–3. This is the node cname (in config sets).
Accelerator or GPU	c*X*-*Y*c*C*s*S*n*N*a*A*	Accelerator or GPU, where *A* = 0.
Aries ASIC	c*X*-*Y*c*C*s*S*a*A*	Cray Aries ASIC on a blade, where *A* is always 0.
Link	c*X*-*Y*c*C*s*S*a*A*l*RC*	Link port of an Aries ASIC, where *RC* = 00–57 (octal). The first digit is the link or LCB row (0–5), and the second digit is the link or LCB column (0–7).

The tree of the major named components looks like the following. Other components may exist and be named in the hierarchy, but typically these are the only ones that matter.

```
s0 (or p0)
|--- cX-Y (cabinet)
     |--- cX-YcC (chassis: 0-2)
          |--- cX-YcCsS (slot: 0-15 (Aries))
               |--- cX-YcCsSnN (node: 0-3)
                    |--- cX-YcCsSnNaA (accelerator/GPU: 0)
               |--- cX-YcCsSaA (Aries: 0)
                    |--- cX-YcCsSaAlRC (Link or LCB Row (0-5)/Column (0-7))
```

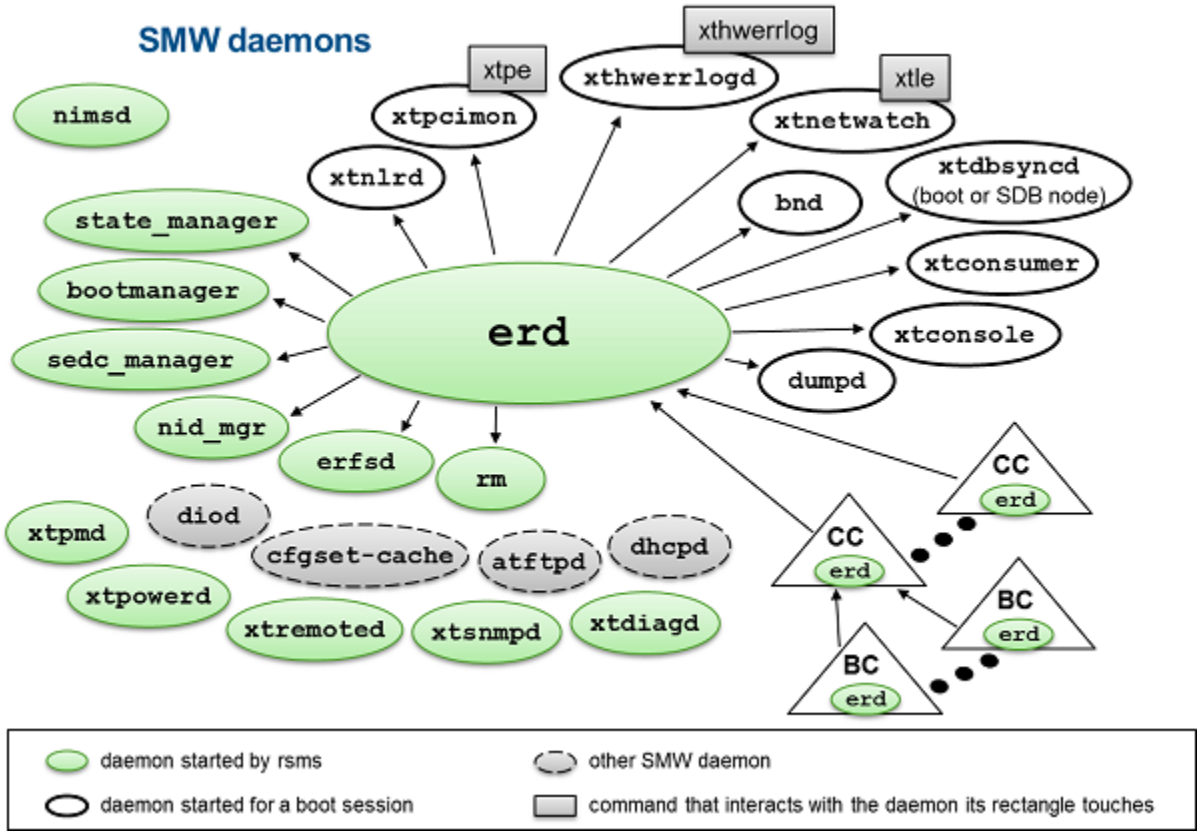
SMW Daemons, Processes, and Logs

- *Daemons on a Stand-alone SMW* on page 13: lists and describes daemons that are started by rsms and should always be running, daemons that are started only during boot, daemons that distribute config set data and regenerate the config set cache, and daemons that are necessary for a PXE boot.
- *Daemons on an SMW HA System* on page 17: describes how to tell whether HA has been installed and configured, and describes daemons and log files that are HA-specific.
- *SMW Log File Locations* on page 19: lists the five kinds of log files and where they are found.
- *Time Synchronization Among XC™ Series System Components* on page 21: describes how time is synchronized among XC components, how to check/set time, and how to query NTP to find information about the time synchronization of a node with its server and peers.

Daemons on a Stand-alone SMW

A variety of daemons are involved in booting an XC™ Series system, as shown in this figure.

Figure 5. Daemons on a Stand-alone SMW



Daemons started by RSMS that should always be running

Several of the daemons running on the SMW are started and monitored by the RSMS service (rsms). These daemons should always be running on the SMW: erd, state_manager, bootmanager, rm, sedc_manager, nid_mgr, erfsd, nimsd, xtpmd, xtpowerd, xtsnmpd, xtremoted, xtdiagd.

Daemons with diverse roles:

nimsd The Node Image Mapping Service (NIMS) daemon (nimsd) maps a node to *boot attributes* that are used when the node is booted.

erd The event router daemon (erd) brokers messages from many software components representing different parts of the XC system. The erd runs on the SMW, but there are also erd running on every blade controller (BC) and cabinet controller (CC), which funnel information back to the erd on the SMW. Other daemons can subscribe to certain types of HSS events to receive notification when those events arrive at the erd.

sm The state_manager (sm) daemon tracks the state of all HSS components (partitions, cabinets, blades, nodes, network ASICs, links, etc.) and stores this in a persistent HSS database—a MariaDB (MySQL) database on the SMW. It responds to queries from `xtcli status`.

bm The bootmanager (bm) daemon coordinates the boot of XC system nodes. bootmanager responds to events from `xtcli boot` and sends events to bcsysd on the BC and the boot node daemon (bnd) on the boot node. It provides for PXE boot of the boot and SDB nodes, high-speed boot over the HSN for all other nodes, or a low-speed boot over the HSS network (for troubleshooting).

rm The router manager (rm) daemon sets up a routing table in the HSN (high speed network).

sedc_manager The system environment data collections manager (sedc_manager) daemon monitors system health and records environmental data for hardware components.

nid_mgr The NID manager (nid_mgr) daemon enables an administrator to control NID assignment and manages the nic-to-nid relationship.

erfsd The event router file system daemon (erfsd) provides persistent storage for the erd on CCs and BCs.

Daemons that are part of power management:

xtpmd The power management daemon (xtpmd) handles all database inserts of node level power data from bcpmd on the BCs and cabinet level power data from ccsysd on the CCs. It also stores ALPS job event data which can be used for offline job-based correlation and energy accounting.

xtpowerd The xtpowerd daemon allows sites to control the rate at which nodes are powered on/off or booted over time to smooth out large fluctuations in power use.

xtsnmpd The Cray SNMP daemon (xtsnmpd) acts as a subagent for snmpd. xtsnmpd responds for two Cray-specific MIBs: CRAY-XC-MIB and CRAY-SMI. The CRAY-SMI MIB provides the structure of management information for the overall Cray enterprise. The CRAY-XC-MIB MIB contains system-level power data, including system-level instantaneous/current power, peak power, average power, and accumulated energy.

xtremoted The xtremoted daemon is the application server which handles CAPMC API requests to connect to the remote database. In most configurations, the remote database is the local PostgreSQL server on the SMW, but it can be configured to send to a remote hostname. See the *XC™ Series Power Management Administration Guide* (S-0043) for more details about a remote database.

xtdiagd The diags daemon (xtdiagd) gets diagnostic data into the Power Management database.

There are two ways to check whether these RSMS daemons are running: see *Check RSMS Daemons* on page 53 in the "Commands Helpful in Troubleshooting a Boot" section.

Daemons that are started only during boot of a CLE system

These daemons are the standard set of processes started only in the context of a CLE boot session.

xtnlrd The network link resiliency daemon (xtnlrd) monitors the blade controllers (BC) for high speed network (HSN) failures and performs recovery actions automatically in the event of a failure. xtnlrd also handles administrative warm swap requests and HSN congestion.

To check whether it is running, see *Check Daemons Using xtalive* on page 56 in the "Commands Helpful in Troubleshooting a Boot" section.

xtpcimon The xtpcimon daemon monitors the health of PCIe (peripheral component interconnect express) channels and logs PCIe link errors to a file.

The `xtpe` command processes pcimon log files for PCIe link errors.

To check for these errors, see *Check for PCIe Link Errors* on page 66 in the "Commands Helpful in Troubleshooting a Boot" section.

xthwerrlogd The xthwerrlogd daemon listens for hardware error events from the ASIC network chip and writes them to a binary file.

The `xthwerrlog` command analyzes that binary file.

To check for these errors, see *Check for Hardware Errors* on page 67 in the "Commands Helpful in Troubleshooting a Boot" section.

xtnetwatch The xtnetwatch daemon monitors the system high-speed network (HSN) faults interconnect for link control block (LCB) and router errors, and it logs them to a file.

The `xtle` command analyzes netwatch log files for HSN errors.

To check for these errors, see *Check for LCB and Router Errors* on page 67 in the "Commands Helpful in Troubleshooting a Boot" section.

xtdbsyncd The xtdbsyncd daemon runs on either the boot (default) or SDB node. The xtdbsyncd daemon synchronizes the SDB to the real hardware status as coordinated by events sent in the hardware supervisory system (HSS). When nodes become available or unavailable, the HSS sends an event, and xtdbsyncd makes the appropriate change to the SDB, designating the node as up or down. Unless a node is designated up, it is not available for allocation by jobs.

bnd When the boot node is up, any requests to boot nodes using the HSN will send a request from the SMW to the boot node daemon (bnd) on the boot node. The boot node NFS-mounts the `/var/opt/cray/imps/boot_images` directory from the SMW to its own `/var/opt/cray/imps/boot_images` mount point so that bnd can access all of the required boot images. bnd will then extract the files from the boot image and initiate a transfer to node memory for each node in the boot request. bnd will report how many nodes in a request succeeded or failed. Output from bnd is sent to the SMW and appears in `/var/opt/cray/log/p0-current/messages-`*`YYYYMMDD`*.

To check whether it is running, see *Check Daemons Using xtalive* on page 56 in the "Commands Helpful in Troubleshooting a Boot" section.

xtconsumer The xtconsumer daemon monitors the erd for HSS events, and its output is redirected to the consumer log file in the directory for the boot session. The `xtconsumer` command monitors the erd to display HSS events as they occur in real time.

xtconsole The xtconsole daemon monitors the erd for console messages, and its output is redirected to the console log file in the directory for the boot session.The `xtconsole` command operates in a shell window and monitors the erd for console messages. `xtconsole` can monitor a single node or multiple nodes. For an example of how to use this command, see *Check Console Messages* on page 55 in the "Commands Helpful in Troubleshooting a Boot" section.

Other SMW daemons

diod The distributed I/O daemon (diod) is started by cray-ids-service, which is a systemd service on the SMW. The diod daemon does I/O forwarding for the IMPS Distribution Service (IDS). Together, cray-ids-service and diod distribute the config set data to nodes on the XC system.

To check whether it is running, see *Check diod daemon* on page 53 in the "Commands Helpful in Troubleshooting a Boot" section.

cfgset-cache The cfgset-cache daemon actively monitors config sets for changes on the SMW. Changes to config sets are processed together, and a SquashFS and associated checksum is generated in response to one or more changes after a period of four seconds elapses with no further change. CLE nodes use the associated SquashFS archives and checksums to ensure that configuration is current. New configuration changes are pulled locally and applied to the CLE node when cray-ansible is run on the node.

To check whether it is running, see *Check cray-cfgset-cache Daemon* on page 54 in the "Commands Helpful in Troubleshooting a Boot" section.

dhcpd, atftpd The Dynamic Host Configuration Protocol (DHCP) server daemon (dhcpd) and the Trivial File Transfer Protocol (TFTP) server daemon (atftpd) are required to PXE boot the boot and SDB nodes.

To check whether they are running, see *Check DHCP or TFTP Daemons* on page 54 in the "Commands Helpful in Troubleshooting a Boot" section.

The boot and SDB nodes PXE boot over the admin network (SMW eth3 10.3.0.0 network). The boot and SDB nodes get `/opt/tftpboot/elilo.efi` and then *`$CNAME`*`.conf`, `bzImage`, and `initramfs.gz` from the *`$CNAME`* directory `/opt/tftpboot/elilo.config/`*`$CNAME`*.

The HSS controllers boot over the HSS network (SMW eth1 10.1.0.0 network). The cabinet controller (CC) gets its boot image from the SMW (`/opt/tftpboot`) and caches it in `/var/tftp`. The CC receives a temporary IP address and the path (relative to `/opt/tftpboot`) for its boot image from the DHCP daemon (dhcpd) on the SMW, whose configuration is in `/etc/dhcpd.conf`.

The blade controller (BC) boots from its CC with the image served up from `/var/tftp` on the CC. The BC receives its temporary IP address and the path (relative to `/var/tftp`) for its boot image from the DHCP daemon (dhcpd) on the CC, whose configuration files are in `/var/etc/udhcpd.conf-eth*`.

Daemons on an SMW HA System

On an SMW HA pair of SMWs, the daemons and processes described for a stand-alone SMW are running only on the first/primary SMW in the cluster of two SMWs. The cluster resource manager (CRM) ensures that these essential daemons are running on the first SMW and not running on the second SMW.

How to determine whether SMW HA is installed and configured

To determine if SMW HA software is installed, use one of these methods:

- Method 1:

```
smw# rpm -q cray-ha-smw
cray-ha-smw-12.0.0.52.geeebae8-2.26.noarch
```

- Method 2:

```
smw# cat /etc/opt/cray/release/smwha-release
RELEASE=12.0.UP00
BUILD=12.0.48
DATE=201605180109
ARCH=x86_64
```

To determine if SMW HA is configured and running properly, use this command (this example shows the output for an HA pair "minnie" and "mickey"):

```
smw# /opt/cray/ha-smw/default/sbin/ha_health

Cluster State
-----------------------------------------
Health State                : Healthy
Active Node                 : minnie
Node-1                      : mickey (online)
Node-2                      : minnie (online)
Number of Resources         : 33
Number of Resources Running : 33
Number of Resources Stopped : 0
Maintenance Mode            : disabled
Stonith Mode                : enabled
```

Indicators in the output that all is well:

- A healthy state
- Both SMWs online
- No resources stopped
- Maintenance mode disabled
- STONITH mode enabled

Note that the number of started resources may be version-specific.

To determine if the distributed replicated block device (DRBD) is healthy, examine `/proc/drbd`.

```
smw# cat /proc/drbd
version: 8.4.4 (api:1/proto:86-101)
GIT-hash: 3c1f46cb19993f98b22fdf7e18958c21ad75176d build by SuSE Build Service
 0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
  ns:905180 nr:0 dw:905772 dr:12762 al:51 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Indicators in the output that all is well:

- A connect state (cs) of Connected
- A data state (ds) of UpToDate/UpToDate

To check the cluster status, use this command on either of the SMWs (this example shows the output for an HA pair "minnie" and "mickey"):

```
smw# crm_mon -1r
Last updated: Tue Sep  6 11:26:06 2016
Last change: Tue Sep  6 08:52:13 2016
Stack: corosync
Current DC: minnie (167903490) - partition with quorum
Version: 1.1.12-ad083a8
2 Nodes configured
33 Resources configured

Online: [ mickey minnie ]

Full list of resources:

 ClusterIP      (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP1     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP2     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP3     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP4     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterMonitor (ocf::smw:ClusterMonitor):      Started minnie
 ClusterTimeSync        (ocf::smw:ClusterTimeSync):     Started minnie
 HSSDaemonMonitor       (ocf::smw:HSSDaemonMonitor):    Started minnie
 Notification   (ocf::heartbeat:MailTo):        Started minnie
 ResourceInit   (ocf::smw:ResourceInit):        Started minnie
 cray-cfgset-cache      (systemd:cray-cfgset-cache):    Started minnie
 dhcpd  (systemd:dhcpd.service):        Started minnie
 fsync  (ocf::smw:fsync):       Started minnie
 hss-daemons    (lsb:rsms):     Started minnie
 stonith-1      (stonith:external/ipmi):        Started mickey
 stonith-2      (stonith:external/ipmi):        Started minnie
 Resource Group: HSSGroup
     postgresqld        (lsb:postgresql):       Started minnie
     mysqld     (ocf::heartbeat:mysql): Started minnie
 Resource Group: IMPSGroup
     cray-ids-service   (systemd:cray-ids-service):     Started minnie
     cray-ansible       (systemd:cray-ansible): Started minnie
     IMPSFilesystemConfig       (ocf::smw:FileSystemConfig):    Started minnie
 Resource Group: LogGroup
     rsyslog    (systemd:rsyslog.service):      Started minnie
     cray-syslog        (systemd:llmrd.service):        Started minnie
     LogFilesystemConfig        (ocf::smw:FileSystemConfig):    Started minnie
 Resource Group: SharedFilesystemGroup
     homedir    (ocf::heartbeat:Filesystem):    Started minnie
     md-fs      (ocf::heartbeat:Filesystem):    Started minnie
     imps-fs    (ocf::heartbeat:Filesystem):    Started minnie
     ml-fs      (ocf::heartbeat:Filesystem):    Started minnie
```

```
    repos-fs    (ocf::heartbeat:Filesystem):    Started minnie
    pm-fs       (ocf::heartbeat:Filesystem):    Started minnie
ip-drbd-pgsql   (ocf::heartbeat:IPaddr2):       Started minnie
Master/Slave Set: ms-drbd-pgsql [drbd-pgsql]
    Masters: [ minnie ]
    Slaves: [ mickey ]
```

Note that the `crm_mon` output displays the SMW host names as "Online" in alphanumeric order; the first SMW shown is not necessarily the primary SMW.

Additional daemons on an SMW HA system

These additional daemons run on both SMWs in an SMW HA system.

- `/usr/sbin/pacemakerd`
- `/usr/lib64/pacemaker/cib`: cluster information base daemon
- `/usr/lib64/pacemaker/stonithd`: "shoot the other node in the head" daemon
- `/usr/lib64/pacemaker/lrmd`: local resource manager daemon
- `/usr/lib64/pacemaker/attrd`:
- `/usr/lib64/pacemaker/pengine`: policy engine daemon
- `/usr/lib64/pacemaker/crmd`: cluster resource manager daemon

These are all part of the SUSE distribution, so for information about them, visit the SUSE Documentation website at *https://www.suse.com/documentation/*. To locate the SUSE high availability guide, select *SUSE Linux Enterprise Server 12*, then search for "sle-ha."

Additional logs on an SMW HA system

In addition to the log files listed in *SMW Log File Locations* on page 19, there are two log files that exist only on SMW HA systems. Note that these are NOT shared between SMWs, so copies from both SMWs may be useful for trouble-shooting purposes.

- `/var/log/pacemaker.log`
- `/var/log/smwha.log`

SMW Log File Locations

There are five kinds of log files:

system-wide The system-wide logs are from the daemons started by the rsms script.

Location SMW

Path `/var/opt/cray/log`

Files

- bootmanager: `bm.out` and `bm.out.1` (previous)
- erd: `event-`*`YYYYMMDD`*
- nid_manager: `nm.out` and `nm.out.1` (previous)

- nimsd: `nimsd.out` and `nimsd.out.1` (previous)
- sedc_manager: `sedc_manager.out` and `sedc_manager.out.1` (previous)
- state_manager: `sm.out` and `sm.out.1` (previous)
- xtdiagd: `xtdiagd.out` and `xtdiagd.out.1` (previous)
- xtpmd: `pmd.out` and `pmd.out.1` (previous)
- xtpowerd: `xtpowerd.out` and `xtpowerd.out.1` (previous)
- xtremoted: `xtremoted.out` and `xtremote.out.1` (previous)
- xtsnmpd: `xtsnmpd.out` and `xtnsnmpd.out.1` (previous)

boot session-specific

The per boot session logs are from the processes started by `xtbootsys` and are therefore also per partition.

Location SMW

Path `/var/opt/cray/log/`*`session-ID`*, where *`session-ID`* has format p*N*-*YYYYMMDD*t*hhmmss* (e.g., p3-20111003t104331)

Files

- xtbootsys log: `bootinfo.`*`session`*
- xtconsole: `console-`*`YYYYMMDD`*
- xtconsumer: `consumer-`*`YYYYMMDD`*
- xthwerrlogd: `hwerrlog.`*`session`*
- xtnetwatch: `netwatch-`*`YYYYMMDD`*
- xtpcimon: `pcimon-`*`YYYYMMDD`*
- xtnlrd: `nlrd.`*`session`*

NOTE: Logs that are written to a *`YYYYMMDD`* files also have a corresponding `.`*`session`* file, but this contains only stdout/stderr from the process and is typically empty.

CC and BC

The logs from the cabinet controllers (CC) and blade controllers (BC) are available on the controller but also are transmitted to the SMW.

Location CC/BC and SMW

Path CC/BC: `/var/log`

SMW: `/var/opt/cray/log/controller`

Files On the SMW, under the `/var/opt/cray/log/controller` directory is a directory for each cabinet in the XC system. Under each cabinet directory is a directory for the CC and each BC in that cabinet, which is where these files are found:

- `messages-`*`YYYYMMDD`*
- `bios-n[0,1,2,3]-`*`YYYYMMDD`*

commands executed on SMW

Logs from commands executed on the SMW are in two locations.

Location	SMW
Path	Output from running `xtdiscover`: `/var/opt/cray/log/xtdiscover` Output from running other commands: `/var/opt/cray/log/commands`

systemd

With SLES12, use the `journalctl` command on the SMW to display the systemd log files, which contain all kernel messages and other available information from systemd..

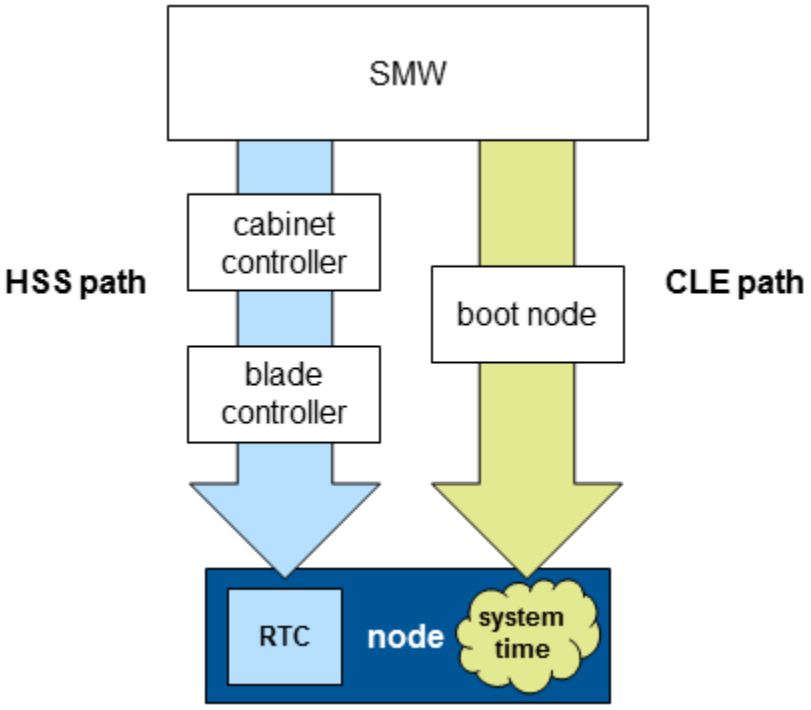
```
smw# journalctl -a
```

Time Synchronization Among XC™ Series System Components

Keeping the clock time synchronized among all the components of a system is important for troubleshooting a failed boot. Log messages are written from several different nodes to the various system logs. Those messages must have accurate and consistent timestamps to make it possible to correlate events on the system.

A Cray XC™ Series system uses the Network Time Protocol (NTP) to achieve synchronization among its components. The SMW is the primary time server, and it provides the reference time for the Hardware Supervisory System (HSS) and for CLE. Both HSS and CLE use the NTP daemon (nptd). For each node in the system, there are two paths to get time from the SMW: the HSS path (SMW → cabinet controller → blade controller → node real-time clock) and the CLE path (SMW → boot node → all other nodes in system). When a node starts up, its real-time clock (RTC), which is a hardware clock, takes its time from the source on the HSS path—the RTC of its blade controller. The system time, which is a software clock, is initially set to the node's RTC, which it obtained from the blade controller. To maintain the system time the node starts the NTP daemon (ntpd) and queries the source on the CLE path—the boot node.

Figure 6. Time Synchronization Among XC™ System Components



Check the time on a node

To access the time on a node, use these commands, which check the time without changing it. There is no change to the node.

- To check the real-time clock (RTC) time:

```
node# hwclock
```

- To check system time:

```
node# date
```

Query NTP

To query NTP, use the `ntpq` program, which can be invoked interactively. To see a list of `ntpq` commands:

```
node# ntpq
ntpq> ?
```

To display help about a command:

```
node# ntpq
ntpq> ? rl
function: read the system or peer variables included in the variable list
usage: rl [ assocID ]
```

Use ntpq to find out if the ntpd server thinks it is synchronized

Use the `rl` command with `ntpq` to determine whether the ntpd server thinks it is synchronized or not. Here is an example:

```
node# ntpq
ntpq> rl
associd=0 status=0615 leap_none, sync_ntp, 1 event, clock_sync,
version="ntpd 4.2.6p5@1.2349-o Tue Apr 28 11:49:15 UTC 2015 (1)",
processor="x86_64", system="Linux/3.12.51-52.39.1_2.2-cray_ari_s", leap=00,
stratum=5, precision=-23, rootdelay=21.255, rootdisp=141.444,
refid=10.3.1.1,
reftime=db1e83d9.949faf83  Wed, Jun 29 2016 12:28:57.580,
clock=db1e8b71.78070fa2  Wed, Jun 29 2016 13:01:21.468, peer=39750,
tc=10, mintc=3, offset=-2.270, frequency=-27.480, sys_jitter=0.000,
clk_jitter=0.913, clk_wander=0.253
```

The leap variable (shown in bold in the example output) indicates one of four things:

00 = no leap second is expected yet
01 = add a second
10 = subtract a second
11 = the ntpd server is currently unsynchronized

Use ntpq to list information about a node's peers

In this example, the command is used on a login node to list information about a node's peers.

```
login# ntpq
ntpq> peers

remote   refid      st t  when poll  reach  delay   offset  jitter
==============================================
*boot    10.3.1.1   5  u  675  1024  377    0.134   0.841   1.183
```

What the columns mean:

remote Peers of the node, specified in the `ntp.conf` file.

* = current time source
# = source selected, distance exceeds maximum value
o = source selected, Pulse Per Second (PPS) used
+ = source selected, included in final set
x = source false ticker
. = source selected from end of candidate list
– = source discarded by cluster algorithm
blank = source discarded high stratum, failed sanity

refid The remote source's synchronization source. (In the example, the source is the SMW, which has IP address 10.3.1.1.)

st Stratum level of the source.

t Types available.

l = local (such as a GPS, WWVB)
u = unicast (most common)
m = multicast
b = broadcast
– = netaddr

when Number of seconds passed since last response

poll Polling interval for source, in seconds.

reach An 8-bit left shift octal value that indicates success/failure to reach source. Success means the bit is set, failure means the bit is not set. The highest value is 377. (In the example, reach = 377, indicating that all were successful.)

delay The round-trip time to receive a reply, in milliseconds.

offset Time difference between the client-server and the source, in milliseconds.

jitter The difference between two time samples, in milliseconds.

Jitter Concerns

Non-application software, including ntpd, running on a node can use up the node's resources and add "jitter" to an application running on the node. This jitter is just the delay caused by the application contending with non-application software for resources like CPUs. This jitter, which will be referred to as application jitter, is different than the jitter mentioned by ntpd. The two should not be confused. On compute nodes, application jitter is bad because it can randomly add some delay to applications running on each node. Parallel programs wait for all nodes to arrive at a barrier before proceeding, so random delays to any node can cause all nodes to wait and waste time.

The NTP daemon (ntpd), which runs on both service and compute nodes, has a polling interval that can be configured to reduce application jitter. The longer the interval, the less jitter is introduced. A short interval (about 10 seconds) works well on service nodes, while compute nodes perform better using a longer polling interval (about 15 minutes).

About Cray Scalable Services

Cray Scalable Services is an essential part of the Cray Management System that is used to both distribute and aggregate information. Within Cray Scalable Services, nodes are designated as SoA (server of authority), tier1, tier2, or tier3. A node can be a member of only one of these groups. Tier1 nodes are clients of the SoA and servers for tier2 nodes. Tier2 nodes are clients of tier1 nodes and servers for tier3 nodes. Tier3 nodes are clients of tier2 nodes. Configuration of nodes as SoA, tier1, and tier2 is defined in the cray_scalable_services configuration service, which must be configured properly for the system to function.

The SMW is the designated SoA in Cray XC systems. The boot and SDB nodes are designated tier1 nodes, and they must have direct network connectivity to the SMW via Ethernet. Typically, tier2 nodes are service nodes or repurposed compute nodes that have no other duties beyond being part of the Scalable Services. All other nodes are tier3 nodes.

This table shows what gets distributed or aggregated using Cray Scalable Services.

from SMW to rest of system	• config set data is shared using a 9P file system and DIOD (distributed I/O daemon) • zypper software repositories can by used from any node with the Live Update feature (http forwarding from the SMW through the tiers)
from boot node to rest of system	• PE (Programming Environment) image root • diag (online diagnostics) image root • Netroot image roots[1]
from rest of system to SMW	• Lightweight Logging Manager (LLM) logging

Here is an example of how Scalable Services works with Live Updates to distribute software out to nodes. Any tier3 node can run zypper to access the repositories on the SMW because it has an entry in `/etc/zypp/repos.d/liveupdates.repo` that points to the tier2 nodes by means of a baseurl, which uses http protocol listing all of the tier2 nodes. The tier2 nodes, in turn, have an entry in `/etc/zypp/repos.d/liveupdates.repo` that lists at least one tier1 node. All tier1 nodes have an entry in `/etc/zypp/repos.d/liveupdates.repo` that lists the SMW.

Services that Depend on Cray Scalable Services

It is important to configure Cray Scalable Services correctly. The following features and services use data from the cray_scalable_services configuration service, and may they not be functional if cray_scalable_services is configured incorrectly.

Node Image Mapping Service (NIMS) plugin	Uses cray_scalable_services data to determine tier1 servers and adds the tier1 kernel command line parameter to each tier1 server.

[1] Netroot is a mechanism that enables nodes booted with a minimal, local in-memory file system to execute within the context of a larger, full-featured root file system which available to the node via a network mount.

IMPS Distribution Service (IDS)	Uses cray_scalable_services data to set the ids kernel command line parameter to the node's parent, from whom it will receive config set data.
DVS Ansible configuration	Uses cray_scalable_services data to determine which nodes should serve DVS file systems. This will also impact Netroot functionality, which uses DVS.
CLE liveupdates functionality	Configured using cray_scalable_services data to determine the parent each node should contact en route to the package repos stored on the SMW.
LLM Ansible configuration	Uses cray_scalable_services data to determine the next server to which a node should send its log data, which depends on the node's tier.
NFS Ansible configuration	Uses cray_scalable_services data to determine which nodes should act as clients and servers.
IP forwarding Ansible configuration	Uses cray_scalable_services data to enable IP forwarding and configure servers' routes depending on their tier.

Anatomy of an XC System Boot with xtbootsys

`xtbootsys` is a large Tcl script used by the crayadm account to boot and shut down CLE on the nodes in the XC system. `xtbootsys` calls several external programs to gather information and run specific commands.

A system administrator can adjust the behavior of `xtbootsys` by using a boot automation file. The `auto.generic` boot automation file is described here. The `auto.generic` and `auto.xtshutdown` files provided by Cray may be changed during a Cray software update or patch. During an initial installation, these files should be copied to site versions such as `auto.`*`hostname`*`.start` and `auto.`*`hostname`*`.stop` (where *`hostname`* is the host name of the XC system) so that they can be customized.

Typical usage for booting is:

```
crayadm@smw> xtbootsys -a auto.hostname.start
```

Typical usage for shutdown is:

```
crayadm@smw> xtbootsys -s last -a auto.hostname.stop
```

The `xtbootsys` command can also stop (`--stop-daemons`), start (`--start-daemons`) and restart (`--restart-daemons`) the background processes that are started by `xtbootsys` at boot time.

The default boot automation files provided by Cray are in `/opt/cray/hss/default/etc`. Beginning with CLE 6.0.UP03, service and compute nodes can boot at the same time. For further information about customizing boot automation files and default boot order, see *About Boot Automation Files* on page 43.

Tasks done by `xtbootsys`

Steps done by `xtbootsys` include calling external programs. These external program might hang, causing a timeout or failure during the booting process. Where an external program is called, dependencies of that program are highlighted.

Here are the tasks done during a boot. Once the boot is complete, a summary will be added to the `bootinfo` log file with the names and duration of these tasks. When analyzing a failed boot, consult the `bootinfo` file for that failed boot session. The order of tasks may vary or there may be some different tasks due to newer software.

1. `initialization`

   a. Create a boot session identifier.

      This boot session identifier is of the form p0-20160621t214422, that is, the partition name followed by the date and time. Notice that there is a "t" character separating the date from the time.

      ```
      ######################################################################
      Your boot session identifier is p0-20160621t214422
      ######################################################################
      ```

   b. Create `bootinfo` log file for this boot session.

The `bootinfo` log file is created underneath the `/var/opt/cray/log` directory in a directory named for the boot session identifier. This log file captures all of the output sent to `stdout` for the `xtbootsys` command, but it also contains information not displayed to `stdout`, such as boot time statistics and details about the md5 checksums on the files in the boot images.

```
#######################################################################
Your boot information will be in /var/opt/cray/log/p0-20160621t214422
#######################################################################
```

c. Record which boot automation file has been used for this boot.

Most boot automation files are in `/opt/cray/hss/default/etc/auto.*`, but they can be in other locations.

d. Check whether the boot manager is running.

2. `xtcli_part_cfg_show`

Check the partition configuration. This step runs `xtcli_part_cfg_show p0` for the p0 partition, which lists the partition, members of the partition, who the boot and SDB nodes are, and all of the boot images required to boot this partition (as assigned via the NIMS daemon.)

```
Your partition configuration is as follows:
Network topology: class 0
=== part_cfg ===
------------------
[partition]: p0: enable (noflags|)
[members]: c0-0
[boot]: c0-0c0s0n1:halt,c0-0c1s0n1:halt
[sdb]: c0-0c0s1n1:halt,c0-0c1s1n1:halt
[NIMS_image 0]: /var/opt/cray/imps/boot_images/initrd-compute-
large_cle_6.0.UP01-build6.0.96_sles_12-created20160615.cpio
[NIMS_image 1]: /var/opt/cray/imps/boot_images/service_cle_6.0.UP01-
build6.0.96_sles_12-created20160614.cpio
[NIMS_image 2]: /var/opt/cray/imps/boot_images/dal_cle_6.0.UP01-
build6.0.96_centos_6.5-created20160614.cpio
[NIMS_image 3]: /var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
[NIMS_image 4]: /var/opt/cray/imps/boot_images/fio-service_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
```

NOTE: This command requires that the state_manager and nimsd daemons be running.

3. `user_input`

In a default boot automation file, the root password is collected for later use in the boot session. Some sites disable this interactive step by setting a variable in the boot automation file with the password.

4. `analyze_archive`

For each boot image, the debug information is extracted from the cpio-formatted file and checksums are calculated with md5sum for each file. Note that only a portion of the information is shown in this example.

```
#######################################################################
Please be patient while I collect additional system information ...
#######################################################################
#######################################################################
cpio -itv -F /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
'cpio -itv -F /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
```

```
build6.0.96_sles_12-created20160615.cpio' completed with status 0
drwxrwxrwt   3 root     root            0 Jun 15 15:07 .
lrwxrwxrwx   1 root     root           12 Jun 15 15:07 initrd-compute-
large_cle_6.0.UP01-build6.0.96_sles_12-created20160615.load -> DEFAULT.load
lrwxrwxrwx   1 root     root            7 Jun 15 15:07 initrd-compute-
large_cle_6.0.UP01-build6.0.96_sles_12-created20160615 -> DEFAULT
-rw-r--r--   1 root     root          398 Jun 15 15:07 DEFAULT.load
-rw-r--r--   1 root     root            0 Jun 15 15:06 imps_image
drwxr-xr-x   3 root     root            0 Jun 15 15:07 DEFAULT
-rw-r--r--   1 root     root     213632219 Jun 15 15:06 DEFAULT/initramfs.gz
-rw-r--r--   1 root     root          273 Jun 15 15:06 DEFAULT/package.info
drwxr-xr-x   4 root     root            0 Jun 15 15:06 DEFAULT/debug
drwxr-xr-x   2 root     root            0 Jun 15 15:06 DEFAULT/debug/boot
-rw-r--r--   1 root     root     100456784 May 18 12:25 DEFAULT/debug/boot/
vmlinux
...
-rw-r--r--   1 root     root      4263504 May 18 12:25 DEFAULT/
bzImage-3.12.51-52.31.1_1.0600.9146-cray_ari_c1253859 blocks
cpio -idmu -F /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
'cpio -idmu -F /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio' completed with status 0
IMPS image detected.
IMPS image detected.
Following are the md5sum values for your cpio archive:
md5sum ./DEFAULT.load
'md5sum ./DEFAULT.load' completed with status 0
6b2da3b30e47c74e2e998142f4c47897  ./DEFAULT.load
md5sum ./imps_image
'md5sum ./imps_image' completed with status 0
d41d8cd98f00b204e9800998ecf8427e  ./imps_image
md5sum ./DEFAULT/initramfs.gz
...
```

5. `xtcli_status_a`

   Show the current status of all components in the system using the `xtcli status -a s0` command. Note that this example was simplified to show a single blade with two service nodes and a single compute blade with four compute nodes. It shows the cabinet controller (CC) c0-0, the blade controller (BC) c0-0c0s0 and c0-0c0s6, the aries on the blade c0-0c0s0a0 and c0-0c0s6a0, and the nodes on the blades c0-0c0s0n1, c0-0c0s0n2, and also c0-0c0s6n0, c0-0c0s6n1, c0-0c0s6n2, c0-0c0s6n3. Notice that on this service blade node c0-0c0s0n0 and c0-0c0s0n3 are marked as empty because there are no node 0 and node 3 on this service blade. Also, node c0-0c0s6n3 is disabled.

```
xtcli status -a s0
Network topology: class 2
Network type: Aries
           Nodeid: Service  Core Arch|  Comp state      [Flags]
-----------------------------------------------------------------------
             c0-0:       -           |       ready      [noflags|]
         c0-0c0s0: service        X86|       ready      [noflags|]
       c0-0c0s0a0: service        X86|          on      [noflags|]
       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|]
...
         c0-0c0s6:       -        X86|       ready      [noflags|]
       c0-0c0s6a0:       -        X86|          on      [noflags|]
       c0-0c0s6n0:       -  IV20  X86|          on      [noflags|]
```

```
       c0-0c0s6n1:       -  IV20  X86|          on      [noflags|]
       c0-0c0s6n2:       -  IV20  X86|          on      [noflags|]
       c0-0c0s6n3:       -  IV20  X86|    disabled      [noflags|]
...
```

NOTE: This command requires that the state_manager daemon be running.

6. `xtcli_status_lcb`

Show the current status of all aries_lcb components in the system using the `xtcli_status_lcb p0` command. Note that the link control blocks are shown as l00 through l57 for a single Aries (the first character in "l00" and "l57" is the letter "l" for link, not the digit "1" for one.)

```
xtcli status -t aries_lcb p0
Network topology: class 2
Network type: Aries
           Nodeid: Service  Core Arch|  Comp state      [Flags]
----------------------------------------------------------------------
    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|         off      [noflags|]
    c0-0c0s0a0l05: service        X86|         off      [noflags|]
    c0-0c0s0a0l06: service        X86|         off      [noflags|]
    c0-0c0s0a0l07: service        X86|         off      [noflags|]
    c0-0c0s0a0l10: service        X86|          on      [noflags|]
    c0-0c0s0a0l11: service        X86|          on      [noflags|]
    c0-0c0s0a0l12: service        X86|          on      [noflags|]
    c0-0c0s0a0l13: service        X86|         off      [noflags|]
    c0-0c0s0a0l14: service        X86|         off      [noflags|]
...
    c0-0c0s0a0l54: service        X86|         off      [noflags|]
    c0-0c0s0a0l55: service        X86|         off      [noflags|]
    c0-0c0s0a0l56: service        X86|         off      [noflags|]
    c0-0c0s0a0l57: service        X86|         off      [noflags|]
    c0-0c0s1a0l00: service        X86|          on      [noflags|]
...
```

NOTE: This command requires that the state_manager daemon be running.

7. `verify_nodelists`

Check the list of nodes.

8. `clean_up_old_daemons`

Remove any daemons associated with a previous boot session.

9. Internal

This step creates the link from `/var/opt/cray/log/p0-current` to `/var/opt/cray/log/`*`SESSIONID`*, for the $*`SESSIONID`* of this boot session.

10. `disable_flood_control`

During the boot process, several nodes will start at the same time and may appear to flood the logging system with console messages. However, this can be tolerated during the booting process and is desirable so that console log messages from the boot can be viewed. It will be enabled later in the booting process so that a console flood after boot time can be handled differently.

```
xtdaemonconfig --partition p0 --type L0 --daemon ER p0 flood_control=0
xtdaemonconfig --partition p0 --type L1 --daemon ER p0 flood_control=0
```

11. `start_xtconsole`

Start `xtconsole` for this boot session. The log from `xtconsole` will be in `/var/opt/cray/log/p0-current/console-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output. `xtconsole` monitors the erd (event router daemon) for console messages from all nodes.

12. `config_bcsysd`

Set some parameters for the blade controller (BC) system daemon, bcsysd (or l0sysd).

```
xtdaemonconfig --partition p0 --daemon=l0sysd --type=L0 p0
diag_mode=false,error_logging=true
```

13. `config_bcbwtd`

Set some parameters for the bandwidth throttling daemon, bcsysd (or l0sysd), running on the BC.

```
xtdaemonconfig --partition p0 --daemon=l0bwtd --type=L0 p0 diag_mode=false
```

14. `xtbounce`

`xtbounce` is used to initialize blades, nodes, or partitions (including the whole system). It is typically run early by `xtbootsys`, but may also be run stand-alone. The general process is:

1. Gather information about components.
2. Halt the nodes.
3. Power down the nodes.
4. Power down the blades (modules).
5. Power up the blades (modules).
6. Power up the nodes.
7. Initialize the HSN links.

This command will show a summary of how long it took `xtbounce` to run.

> **NOTE:** This process requires interaction with several HSS daemons: State manager (state_manager) on the SMW, NID Manager (nid_mgr) on the SMW, ccsysd on the cabinet controllers, and bcsysd on the blade controllers.
>
> **NOTE:** Some large systems do the `xtbounce` and `rtr` commands before running `xtbootsys` and set variables in their boot automation file so that these two steps are skipped when running `xtbootsys`.
>
> **NOTE:** If a boot fails because `xtbounce` fails, then addressing the problem and running `xtbounce` interactively may be needed. However, if a boot succeeded with `xtbounce` and failed later on, running `xtbounce` interactively will mean that the entire `xtbootsys` will have to be restarted.

```
xtbounce p0
***** get_class *****
21:46:09 - Beginning to wait for response(s)
21:46:09 - Done waiting for the State Manager
***** get_nids *****
21:46:09 - Beginning to wait for response(s)
```

```
21:46:10 - Done waiting for the NID Manager
***** gather_partition_info *****
21:46:10 - Beginning to wait for response(s)
21:46:10 - Done waiting for the State Manager
***** check_partition_info *****
***** gather_user_components *****
21:46:10 - Beginning to wait for response(s)
21:46:11 - Done waiting for the State Manager
***** gather_partition_components *****
***** cross_check *****
INFO: found no accelerators in user-supplied component list
INFO: you will be affecting 1 cabinets, 8 modules, 24 nodes
***** aries_gather_cab_pwr_states *****
21:46:11 - Beginning to wait for response(s)
21:46:11 - Received 1 of 1 responses
INFO: power state checked on 1 cabinets
***** alive *****
21:46:11 - Beginning to wait for response(s)
21:46:12 - Received 8 of 8 responses
***** unload *****
21:46:12 - Beginning to wait for response(s)
21:46:12 - Received 8 of 8 responses
***** aries_download *****
INFO: Executing xtpmaction to apply active power profile
SUCCESS:'/opt/cray/hss/default/etc/pcap.sh p0 /tmp/xtbounce-uQJ9n8/
xtpmaction_node_list' completed with result 0
INFO: creating routing config file 'rtr-cfg.default' in '/tmp/xtbounce-tO3N2F'
INFO: creating AOC routing config file 'rtr-cfg.aoc' in '/tmp/xtbounce-tO3N2F'
INFO: non-default MMR data will not be downloaded
INFO: using DEFAULT Aries NP firmware files from '/opt/cray/serdes/aries/
default/np'
INFO: packing /opt/cray/serdes/aries/default/np/
aries_np_8051_dram_cray_ver_03_0b.txt, 40118 bytes
INFO: packing /opt/cray/serdes/aries/default/np/
aries_np_8051_iram_cray_ver_03_0b.txt, 106496 bytes
INFO: packing /opt/cray/serdes/aries/default/np/np_dram.hex, 40118 bytes
INFO: packing /opt/cray/serdes/aries/default/np/np_fw.hex, 106496 bytes
INFO: No /opt/cray/hss/default/etc/bios_settings file found.
INFO: No /opt/cray/hss/default/etc/phy_cmp_offset file found.
INFO: No /opt/cray/hss/default/etc/pre_nodeup_mmrs file found.
INFO: No /opt/cray/hss/default/etc/snowbush_phy_workaround1.mmrs file found.
Sending: 295784 bytes Total sent so far: 295784 Total to send: 295784
21:46:23 - Beginning to wait for response(s)
21:46:42 - Received 8 of 8 responses
***** halt_node *****
21:46:24 - Beginning to wait for response(s)
21:46:24 - Received 24 of 24 responses
***** node_down *****
21:46:24 - Beginning to wait for response(s)
21:46:25 - Received 0 of 24 response
21:46:31 - Received 24 of 24
responses
***** module_down *****
21:46:31 - Beginning to wait for response(s)
21:46:32 - Received 0 of 8 response
21:46:35 - Received 8 of 8 responses
***** module_up *****
21:46:35 - Beginning to wait for response(s)
21:46:35 - Received 0 of 8 responses
***** aries_node_up *****
21:46:42 - Beginning to wait for response(s)
```

```
21:46:42 - Received 0 of 24 responses
21:50:37 - Received 20 of 24 responses
21:50:37 - Received 24 of 24 responses
***** aries_link_init *****
INFO: Performing fast pre-tuned NP/SerDes init.
21:50:37 - Beginning to wait for response(s)
21:50:38 - Received 0 of 8 responses
21:51:07 - Received 8 of 8 responses
***** aries_link_deadstart *****
21:51:07 - Beginning to wait for response(s)
21:51:08 - Received 6 of 8 responses
21:51:08 - Received 8 of 8 responses
***** link_check *****
21:51:08 - Beginning to wait for response(s)
21:51:09 - Received 8 of 8 responses
INFO: Gathering SDB node info for partition p0
Of the 8 L0s and 24 nodes:
8 L0s were found to be alive
8 modules running HSN links
24 nodes are powered up
INFO: removed /tmp/xtbounce-tO3N2F/rtr-cfg.default
INFO: removed /tmp/xtbounce-tO3N2F/rtr-cfg.aoc
INFO: removed /tmp/xtbounce-tO3N2F
Total runtime: 300.0 seconds
```

15. `cable_check`

Check the HSN cabling information with `xtcablecheck`.

16. `xthwinv`

Prepare hardware inventory information with `xthwinv`.

> **NOTE:** This command requires that the state_manager daemon be running and that a successful `xtbounce` has completed.

17. `xthwinv_X`

Prepare XML-formatted hardware inventory information with `xthwinv -X`.

> **NOTE:** This command requires that the state_manager daemon be running and that a successful `xtbounce` has completed.

18. `xtsdbhwcache`

Mark the hwinv cache generated above as being "clean" for the SDB node to use.

19. `xtclear_alert`

Clear alerts. This ensures that any alerts will be from this booting session.

20. `xtclear_warn`

Clear warnings. This ensures that any warnings will be from this booting session.

21. `route_setup`

Route the system with `rtr -R`. When `rtr` has success, there is no output. If `rtr` fails, there will be output from this command.

> **NOTE:** Some large systems do the `xtbounce` and `rtr` commands before running `xtbootsys` and set variables in their boot automation file so that these two steps are skipped when running `xtbootsys`.

22. `start_xtconsole_1`

Start `xtconsumer` for this boot session watching for ec_node_info messages. The log from `xtconsumer` will be in `/var/opt/cray/log/p0-current/consumer-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output.

23. `start_xtnetwatch`

Start `xtnetwatch` for this boot session. The log from `xtnetwatch` will be in `/var/opt/cray/log/p0-current/netwatch-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output.

24. `start_xtpcimon`

Start `xtpcimon` for this boot session. The log from `xtpcimon` will be in `/var/opt/cray/log/p0-current/pcimon-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output.

This data can be processed with the `xtpe` command.

25. `start_dumpd`

Start `xtdumpd` for this boot session. The log from `xtdumpd` will be in `/var/opt/cray/log/p0-current/dumpd-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output.

26. `start_xthwerrlogd`

Start `xthwerrlogd` for this boot session. The data from `xthwerrlogd` will be in `/var/opt/cray/log/p0-current/hwerrlog.$`*`SESSIONID`*. This data can be viewed using the `xthwerrlog` command.

27. `start_xtnlrd`

Start `xtnlrd` for this boot session. The log from `xtnlrd` will be in `/var/opt/cray/log/p0-current/nlrd-`*`YYYYMMDD`* for the current day of the month in the year. Each day a new file will be created for this output.

28. `start_xtwatcher`

Start `xtwatcher` for this boot session. `xtwatcher` will watch for the daemons started by `xtbootsys` (xtconsole, xtnetwatch, xtconsumer, xthwerrlog, xtnlrd, dumpd, and xtpcimon) and restart any which are no longer running.

29. `start_sec`

Start `cray_sec`. See *XC™ Series SEC Configuration Guide* (S-2542) for details on how to configure the Simple Event Correlator (SEC).

30. `crms_exec` for `nims_liaison.py`

This `crms_exec` task name means that an external program is being called.

Merge IDS settings from the CLE config set into NIMS map for this boot with the hardware components that are currently available.

The `/opt/cray/imps-distribution/default/bin/nims_liaison.py` program will check the Cray scalable services information in the CLE config set to ensure that kernel parameters for each node reflect their position within the hierarchy of SMW to tier1 to tier2 to tier3 nodes. Any node should look towards the SMW to a node in a lower number tier. Nodes will have one to three server IP addresses in their kernel

parameter for IDS. The IDS servers will be used to get config set information for that node during the boot process.

NOTE: Note: This step requires that the NIMS daemon (nimsd) be running on the SMW.

31. `crms_set_failed_option`

This action is in the boot automation file.

If the boot or SDB nodes fail, immediately exit `xtbootsys`. Later this will be changed for other service nodes to pause for user input rather than exiting `xtbootsys`.

32. `crms_set_failed_timeout`

This action is in the boot automation file.

If the boot or SDB nodes time out during their boot, immediately exit `xtbootsys`. Later this timeout value will be changed for other service nodes.

33. `crms_exec_1` for `xtdaemonconfig`

This is the earliest point at which commands from the boot automation file will be run. All of the previous tasks are the same for all XC systems.

For systems with boot node failover or SDB node failover configured, the boot automation file should have lines similar to these, which enable STONITH on the blades containing the primary boot node and the primary SDB node. This example uses blades c0-0c0s0 and c0-0c0s1.

```
lappend actions {crms_exec "xtdaemonconfig c0-0c0s0 stonith=true"}
lappend actions {crms_exec "xtdaemonconfig c0-0c0s1 stonith=true"}
```

34. `boot_bootnode_sdbnode`

The next step in the boot automation file is to boot the boot and SDB nodes. The boot and SDB nodes can be booted via PXE boot at the same time from the SMW (this assumes the SDB image is small enough; if it is too large, SDB nodes will be booted after the boot nodes). They are both tier1 nodes in Cray scalable services, with an Ethernet connection to the SMW. This entry in the boot automation file will run the `xtcli boot` command.

```
# PXE boot both the bootnode and sdbnode together
lappend actions {crms_boot_bootnode_sdbnode}
```

And the command will be shown in the bootinfo log file with all of the boot nodes and SDB nodes for this partition. This example has boot nodes as c0-0c0s0n1 and c0-0c1s0n1 and SDB nodes as c0-0c0s1n1 and c0-0c1s1n1.

```
xtcli -s boot service  -o pxeboot c0-0c0s0n1,c0-0c1s0n1,c0-0c0s1n1,c0-0c1s1n1
```

While these nodes are booting, there should be output for each node to the console log file from the `xtconsole` session started earlier. This output can also be watched by using the `xtconsole` command in another window from where `xtbootsys` is executing.

```
smw# xtconsole -at
```

As the boot and SDB nodes boot, more messages will appear in the `xtbootsys` window indicating the status of the PXE boot.

```
xtcli -s boot service  -o pxeboot c0-0c0s0n1,c0-0c1s0n1,c0-0c0s1n1,c0-0c1s1n1
Network topology: class 0
```

```
 16 % |****                          |  00:00:00
Getting configuration
 33 % |**********                    |  00:00:00
Starting PXE boot
 50 % |***************               |  00:00:00
State manager transition nodes
 66 % |********************          |  00:00:00
Getting NIMS configuration
 83 % |*************************     |  00:00:01
Booting PXE
100 % |******************************|  00:00:13
Boot request done

              Nodeid|       Flags:      Result
----------------------------------------------------------------
          c0-0c0s0n1|    noflags|:      Success
          c0-0c0s1n1|    noflags|:      Success
          c0-0c1s0n1|    noflags|:      Success
          c0-0c1s1n1|    noflags|:      Success
----------------------------------------------------------------
Tue Jun 21 21:51:33 CDT 2016
It took 13 seconds for 'xtcli' to complete.
'xtcli -s boot service  -o pxeboot c0-0c0s0n1,c0-0c0s1n1,c0-0c0s9n2' completed
with status 0
```

35. `extract_debug` (several tasks)

Extract debug information from each of the boot images. After the boot and SDB nodes have begun their boot, debug information is extracted from all boot images using the `xtextractdebug` command.

36. `wait_for_bootnode_sdbnode`

The boot session will wait for the event code which means that the boot and SDB nodes have completed.

```
crms_wait_for_linux_boot: nodelist: c0-0c0s0n1 c0-0c0s1n1 c0-0c1s0n1 c0-0c1s1n1
waiting for 4 nodes to finish booting ...

Tue Jun 21 21:51:58 CDT 2016
While waiting for the 4 nodes to boot, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting
    enter a '?' to see which nodes we're still waiting for
    enter anything else to see how many nodes we're still waiting for
    do nothing and just watch
node 'c0-0c1s0n1' is done (3 left)
node 'c0-0c1s1n1' is done (2 left)
node 'c0-0c0s0n1' is done (1 left)
node 'c0-0c0s1n1' is done (0 left)
Tue Jun 21 21:58:05 CDT 2016
It took 367 seconds (6 minutes, 7 seconds) for 4 node(s) to boot.
```

37. `crms_set_failed_option_1`

This action is in the boot automation file.

Change the setting from the boot and SDB nodes (which would exit `xtbootsys` if they failed) so that other service nodes will "prompt" the user about what action to take.

38. `crms_set_failed_timeout_1`

This action is in the boot automation file.

Change the setting from the boot and SDB nodes (which would exit `xtbootsys` if they timed out during the boot) so that other service nodes will "prompt" the user about what action to take.

39. Boot the rest of the service and compute nodes.

There are two different methods for getting the rest of the service and compute nodes booted, depending on the release and whether direct-attached Lustre (DAL) is used:

Method 1	Boot service and compute nodes at the same time.	For systems running CLE 6.0.UP03 or a later release and do not have DAL. Do NOT use this method if there are other actions to be done after the service nodes complete booting and before the compute nodes begin booting.
Method 2	Boot service nodes first, then boot compute nodes.	For systems running CLE 6.0.UP02 or an earlier release, or systems with DAL, or systems that have other actions to be done after the service nodes complete booting and before the compute nodes begin booting.

Method 1: Boot service and compute nodes at the same time.

a. `boot_all`

Boot all of the remaining service and compute nodes. This action is in the boot automation file.

Since the boot node is up, these nodes will be booted using the HSN via interaction with the boot node daemon (bnd). The boot node NFS mounts the `/var/opt/cray/imps/boot_images` directory from the SMW to its own `/var/opt/cray/imps/boot_images` mount point so that bnd can access all of the required boot images. Notice that the primary boot node and primary SDB node are already in the "ready" state from their previous boot, and the alternate boot node and alternate SDB node also had their boot initiated earlier, so those nodes will not transition to "Success" when checked for their status after the boot.

```
xtcli -s boot all  p0
Network topology: class 0
 14 % |****                          |  00:00:00
Getting configuration
 28 % |********                      |  00:00:00
Starting nodes
 42 % |************                  |  00:00:00
State manager transition nodes
 57 % |*****************             |  00:00:00
Getting NIMS configuration
 71 % |*********************         |  00:00:01
Transferring boot image over HSN
Tue Jun 21 21:58:17 CDT 2016
While waiting for the command to complete, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting (hang up on spawned process)
    enter anything else to repeat this message
    do nothing and just watch
 85 % |*************************     |  00:01:03
Booting requested nodes
100 % |******************************|  00:01:03
Boot request done

              Nodeid|       Flags:      Result
----------------------------------------------------------------
```

```
          c0-0c0s0n1|    noflags|:      Item state 'ready' is invalid for
requested command
          c0-0c0s0n2|    noflags|:      Success
          c0-0c0s1n1|    noflags|:      Item state 'ready' is invalid for
requested command
          c0-0c0s1n2|    noflags|:      Success
          c0-0c1s0n1|    noflags|:      Failover node is in invalid state
for booting
          c0-0c1s0n2|    noflags|:      Success
          c0-0c1s1n1|    noflags|:      Failover node is in invalid state
for booting
          c0-0c1s1n2|    noflags|:      Success
----------------------------------------------------------------
Tue Jun 21 21:59:17 CDT 2016
It took 64 seconds (1 minute, 4 seconds) for 'xtcli' to complete.
```

b. `wait_for_all`

Wait for all service and compute nodes to complete their boot. The boot session will wait for the event code which means that the nodes being booted have completed.

```
crms_wait_for_linux_boot: nodelist: c0-0c0s2n2 c0-0c0s4n1 c0-0c0s4n2
c0-0c0s5n1 c0-0c0s5n2 c0-0c0s8n0 c0-0c0s8n1 c0-0c0s8n2 c0-0c0s8n3
c0-0c0s10n0 c0-0c0s10n1 c0-0c0s10n2 c0-0c0s10n3
waiting for 13 nodes to finish booting ...

Tue Jun 21 21:59:17 CDT 2016
While waiting for the 13 nodes to boot, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting
    enter a '?' to see which nodes we're still waiting for
    enter anything else to see how many nodes we're still waiting for
    do nothing and just watch
node 'c0-0c0s4n1' is done (12 left)
node 'c0-0c0s4n2' is done (11 left)
node 'c0-0c0s5n2' is done (10 left)
node 'c0-0c0s5n1' is done (9 left)
node 'c0-0c0s2n2' is done (8 left)
node 'c0-0c0s10n0' is done (7 left)
node 'c0-0c0s10n3' is done (6 left)
node 'c0-0c0s10n1' is done (5 left)
node 'c0-0c0s10n2' is done (4 left)
node 'c0-0c0s8n0' is done (3 left)
node 'c0-0c0s8n2' is done (2 left)
node 'c0-0c0s8n3' is done (1 left)
node 'c0-0c0s8n1' is done (0 left)
Tue Feb 14 10:00:34 CST 2017
It took 365 seconds (6 minutes, 5 seconds) for 13 node(s) to boot.
```

Method 2: Boot service nodes first, then boot compute nodes.

For systems with DAL or for systems running CLE 6.0.UP02 or an earlier release, the service nodes and compute nodes must be booted in separate steps. If there are other actions to be done after the service nodes complete their boot and before the compute nodes begin their boot, then use this method even for systems running CLE 6.0.UP03 that do not have DAL.

a. `boot_all_serv`

Boot all of the service nodes. This action is in the boot automation file.

Since the boot node is up, these nodes will be booted using the HSN via interaction with the boot node daemon (bnd). The boot node NFS mounts the `/var/opt/cray/imps/boot_images` directory from the SMW to its own `/var/opt/cray/imps/boot_images` mountpoint so that bnd can access all of the required boot images.

Notice that the primary boot node and primary SDB node are already in the "ready" state from their previous boot and the alternate boot node and alternate SDB node also had their boot initiated earlier, so those nodes will not transition to "Success" when checked for their status after the boot.

```
xtcli -s boot all_serv  p0
Network topology: class 0
 14 % |****                        |  00:00:00
Getting configuration
 28 % |********                    |  00:00:00
Starting service nodes
 42 % |************                |  00:00:00
State manager transition nodes
 57 % |*****************           |  00:00:00
Getting NIMS configuration
 71 % |*********************       |  00:00:02
Transferring boot image over HSN
Tue Jun 21 21:58:17 CDT 2016
While waiting for the command to complete, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting (hang up on spawned process)
    enter anything else to repeat this message
    do nothing and just watch
 85 % |*************************   |  00:01:10
Booting service nodes
100 % |******************************|  00:01:11
Boot request done

              Nodeid|       Flags:      Result
-----------------------------------------------------------------
          c0-0c0s0n1|    noflags|:      Item state 'ready' is invalid for
requested command
          c0-0c0s0n2|    noflags|:      Success
          c0-0c0s1n1|    noflags|:      Item state 'ready' is invalid for
requested command
          c0-0c0s1n2|    noflags|:      Success
          c0-0c1s0n1|    noflags|:      Failover node is in invalid state
for booting
          c0-0c1s0n2|    noflags|:      Success
          c0-0c1s1n1|    noflags|:      Failover node is in invalid state
for booting
          c0-0c1s1n2|    noflags|:      Success
-----------------------------------------------------------------
Tue Jun 21 21:59:17 CDT 2016
It took 72 seconds (1 minute, 12 seconds) for 'xtcli' to complete.
```

b. `wait_for_all_serv`

Wait for service nodes to complete their boot. The boot session will wait for the event code which means that the nodes being booted have completed.

```
crms_wait_for_linux_boot: nodelist: c0-0c0s0n2 c0-0c0s1n2 c0-0c1s0n2
c0-0c1s1n2
waiting for 4 nodes to finish booting ...
```

```
Tue Jun 21 21:59:17 CDT 2016
While waiting for the 4 nodes to boot, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting
    enter a '?' to see which nodes we're still waiting for
    enter anything else to see how many nodes we're still waiting for
    do nothing and just watch
node 'c0-0c0s0n2' is done (3 left)
node 'c0-0c0s1n2' is done (2 left)
node 'c0-0c1s0n2' is done (1 left)
node 'c0-0c1s1n2' is done (0 left)
Tue Jun 21 22:16:01 CDT 2016
It took 404 seconds (6 minutes, 44 seconds) for 4 node(s) to boot.
```

c. crms_exec_on_bootnode (several)

Run other boot automation commands after service nodes have booted and before starting to boot compute nodes.

The method shown below for DAL runs two different commands on the boot node as the root username with the `crms_exec_on_bootnode` routine. If commands need to be run on another service node, use the `crms_exec_via_bootnode` routine and then list which host name, username, and command.

See xtbootsys(8) for more information on what can be done in a boot automation file.

1. Start DAL

For systems with DAL (Direct-attached Lustre), the next step in the boot automation file is to start the Lustre server on DAL nodes and then mount the Lustre file system on login nodes.

```
lappend actions { crms_exec_on_bootnode "root" "lustre_control start -f
dal" }
lappend actions { crms_exec_on_bootnode "root" "lustre_control
mount_clients -f dal -w login[1-8]" }
```

Output from each of these commands will be shown and the status from running them on the specified nodes.

d. `boot_all_comp`

Boot all compute nodes.

Since the boot node is up, these nodes will be booted using the HSN via interaction with the boot node daemon (bnd). The boot node NFS mounts the `/var/opt/cray/imps/boot_images` directory from the SMW to its own `/var/opt/cray/imps/boot_images` mountpoint so that bnd can access all of the required boot images.

```
xtcli -s boot all_comp  p0
Network topology: class 0
 14 % |****                        |  00:00:00
Getting configuration
 28 % |********                    |  00:00:00
Starting compute nodes
 42 % |************                |  00:00:00
State manager transition nodes
 57 % |****************            |  00:00:00
Getting NIMS configuration
 71 % |********************        |  00:00:01
Transferring boot image over HSN
Tue Jun 21 22:16:55 CDT 2016
```

```
While waiting for the command to complete, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting (hang up on spawned process)
    enter anything else to repeat this message
    do nothing and just watch
 85 % |************************    |  00:00:13
Booting compute nodes
100 % |****************************|  00:00:13
Boot request done

              Nodeid|       Flags:      Result
----------------------------------------------------------------
          c0-0c0s8n0|    noflags|:      Success
          c0-0c0s8n1|    noflags|:      Success
          c0-0c0s8n2|    noflags|:      Success
          c0-0c0s8n3|    noflags|:      Success
          c0-0c1s9n0|    noflags|:      Success
          c0-0c1s9n1|    noflags|:      Success
          c0-0c1s9n2|    noflags|:      Success
          c0-0c1s9n3|    noflags|:      Success
----------------------------------------------------------------
Tue Jun 21 22:16:58 CDT 2016
It took 14 seconds for 'xtcli' to complete.
```

e. `wait_for_all_comp`

Wait for compute nodes to complete their boot. The boot session will wait for the event code which means that the nodes being booted have completed.

```
crms_wait_for_linux_boot: nodelist: c0-0c0s8n0 c0-0c0s8n1 c0-0c0s8n2
c0-0c0s8n3 c0-0c1s9n0 c0-0c1s9n1 c0-0c1s9n2 c0-0c1s9n3
waiting for 8 nodes to finish booting ...

Tue Jun 21 22:16:58 CDT 2016
While waiting for the 8 nodes to boot, you can ...
    enter a 'q' to quit xtbootsys
    enter a 's' to stop waiting
    enter a '?' to see which nodes we're still waiting for
    enter anything else to see how many nodes we're still waiting for
    do nothing and just watch
node 'c0-0c0s8n0' is done (7 left)
node 'c0-0c0s8n2' is done (6 left)
node 'c0-0c0s8n3' is done (5 left)
node 'c0-0c0s8n1' is done (4 left)
node 'c0-0c1s9n1' is done (3 left)
node 'c0-0c1s9n0' is done (2 left)
node 'c0-0c1s9n2' is done (1 left)
node 'c0-0c1s9n3' is done (0 left)
Tue Jun 21 22:25:12 CDT 2016
It took 493 seconds (6 minutes, 13 seconds) for 8 node(s) to boot.
```

40. Additional boot automation commands

Run any remaining boot automation commands after compute nodes have booted.

If there are any other boot automation commands to be done after the compute nodes have booted, they will be run now.

All of the steps after this point are not in the boot automation.

41. `gather_ko`

Gather kernel object files from boot node. Several `ssh` and `rsync` commands will be run from the SMW to the boot node to collect more information to be stored on the SMW in `/var/opt/cray/debug/$SESSIONID/boot-root`.

```
Gathering ko files from bootnode:/lib/modules/`uname -r`
```

42. `gather_fstab`

This step gathers `/etc/fstab` from one of the compute nodes.

43. `clean_up`

44. `enable_flood_control`

Enable the flood control which was disabled earlier in this boot session. During the boot process, several nodes will start at the same time and may appear to flood the logging system with console messages. However, this can be tolerated during the booting process and is desirable so that console log messages from the boot can be viewed.

```
xtdaemonconfig --partition p0 --type L0 --daemon ER p0 flood_control=1
xtdaemonconfig --partition p0 --type L1 --daemon ER p0 flood_control=1
```

45. Boot session summary

A summary of the boot session will be displayed.

```
This session took 2454 seconds (40 minutes, 54 seconds).

##############################################################
Session Boot Summary:
  14 nodes completed their boot
##############################################################
```

46. Display boot time statistics

Boot time statistics are displayed only to the `bootinfo` log file, but it does show the duration of different parts of the booting process, identified by the name of the `xtbootsys` task. This example shows that the ability to boot all service and compute nodes at the same time can significantly decrease overall system boot time.

```
#####################################################################
# Boot Time Statistics:                                             #
#####################################################################
# TASK                        CONCURRENT DURATION                   #
# initialization                         0m0s                       #
# xtcli_part_cfg_show                    0m4s                       #
# user_input                             0m4s                       #
# analyze_archive                        1m12s                      #
# xtcli_status_a                         0m0s                       #
# xtcli_status_lcb                       0m0s                       #
# verify_nodelists                       0m0s                       #
# clean_up_old_daemons                   0m1s                       #
# Internal                               0m22s                      #
# disable_flood_control                  0m1s                       #
# start_xtconsole                        0m0s                       #
# config_bcsysd                          0m0s                       #
# config_bcbwtd                          0m0s                       #
# xtbounce                               3m53s                      #
# cable_check                            0m1s                       #
# xthwinv                                0m2s                       #
# xthwinv_X                              0m1s                       #
```

```
# xtsdbhwcache                          0m1s                          #
# xtclear_alert                         0m0s                          #
# xtclear_warn                          0m1s                          #
# route_setup                           0m0s                          #
# start_xtconsole_1                     0m0s                          #
# start_xtnetwatch                      0m0s                          #
# start_xtpcimon                        0m0s                          #
# start_dumpd                           0m0s                          #
# start_xthwerrlogd                     0m0s                          #
# start_xtnlrd                          0m0s                          #
# start_xtwatcher                       0m0s                          #
# crms_exec                             0m4s                          #
# crms_set_failed_option                0m0s                          #
# crms_set_failed_timeout               0m0s                          #
# boot_bootnode_sdbnode                 0m22s                         #
# wait_for_bootnode_sdbnode     YES     4m32s                         #
# extract_debug                 YES     0m6s                          #
# extract_debug_1               YES     0m3s                          #
# extract_debug_2               YES     0m5s                          #
# extract_debug_3               YES     0m6s                          #
# crms_set_failed_option_1              0m0s                          #
# crms_set_failed_timeout_1             0m0s                          #
# boot_all                              1m4s                          #
# wait_for_all                          6m5s                          #
# crms_exec_via_bootnode                0m1s                          #
# gather_ko                             0m4s                          #
# gather_fstab                          0m3s                          #
# clean_up                      YES     0m1s                          #
# enable_flood_control          YES     0m1s                          #
# Total                                 17m55s                        #
#######################################################################
```

About Boot Automation Files

New for the CLE 6.0.UP03 release. With this release, the default boot behavior for Cray systems without direct-attached Lustre (DAL) nodes is to boot all service nodes (other than the boot and SDB nodes) and all compute nodes can boot at the same time, thereby decreasing overall boot time.

- Default for systems without DAL:
  1. Boot + SDB (if SDB image small enough to PXE boot)
  2. SDB (if SDB image too large to PXE boot)
  3. Service + Compute
- Default for systems with DAL:
  1. Boot + SDB (if SDB image small enough to PXE boot)
  2. SDB (if SDB image too large to PXE boot)
  3. Service
  4. Compute

Cray provides the following boot automation files with this release.

auto.generic	Used to boot the entire XC system.
auto.xtshutdown	Used to shut down the entire XC system.

auto.bootnode	Used to boot only the boot node(s).
auto.bootnode+sdb	Used to boot only the boot node(s) and SDB node(s).

During a fresh install, sites typically copy `auto.generic`, rename it with the host name of the system for which it will be used (`auto.`*`hostname`*`.start`), and customize it for that site and system. Likewise, sites typically copy `auto.xtshutdown`, rename it with the host name of the system for which it will be used (`auto.`*`hostname`*`.stop`), and customize it, as needed. The host name is included because different systems may have different software installed, resulting in different boot or shutdown requirements. For example, on a system with PBS (a workload manager) installed, extra commands may be needed in the `auto.`*`hostname`*`.stop` file to cleanly stop the PBS queues on SDB or MOM nodes before shutting down the nodes.

When is customization of an automation file needed?

- For systems booting tmpfs images (instead of Netroot) with no SDB node failover, no changes may be necessary.
- For systems booting Netroot images, instructions for making Netroot-related changes after the first boot with tmpfs are provided at the appropriate place in the fresh install process.
- For systems booting direct-attached Lustre (DAL) images, instructions for making DAL-related changes are provided at the appropriate place in the fresh install process.
- For systems with added content in the recipe used for SDB nodes, if the resulting custom recipe produces a boot image too large for a PXE boot, changes to the boot automation file are necessary. If based on `auto.generic`, the system boot automation file will have an option (commented out by default) to boot the boot node via PXE boot and then boot the SDB node via the HSN.
- For systems with a workload manager (WLM) installed, WLM-related changes may be needed. Specific commands to add will vary based on the WLM.

The Booting Process from the CLE Node View

Boot troubleshooting may include examining one or more nodes during the boot, so an understanding of the booting process from the node view may be helpful. The booting process is different depending on whether it is a PXE boot or a boot over the high speed network (HSN), and whether a tmpfs image or Netroot image is being booted. But much of the process is common to all nodes when cray-ansible begins running on the node and writing logs of how Ansible plays consume config set data to personalize the node for its role within the system.

About PXE boot versus HSN boot

Most nodes are booted over the HSN, but there are a few nodes that PXE boot from the SMW because they are nodes with a direct Ethernet connection to the SMW. The boot and SDB nodes are tier1 nodes that PXE boot from the SMW, as described in *Booting with PXE Boot for Boot and SDB Nodes* on page 46.

About booting a tmpfs image versus a Netroot image

The Cray XC™ Series root file system for nodes can either reside in RAM (tmpfs) or be mounted from a network source (Netroot), depending on the type of node. The boot and SDB nodes, all other service nodes (except login nodes), and all DAL (direct-attached Lustre) nodes must use tmpfs. Compute nodes and login nodes may use either tmpfs or Netroot.

To check whether node c0-0c0s0n2 is using Netroot from the SMW, run this command:

```
smw# cnode list c0-0c2s0n1
NAME         TYPE     GROUP
IMAGE
                                        CONFIG_SET  EXT_PARAMETERS
c0-0c0s0n2   service  login
/var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
    -           sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 NIMS_GROUP=login
netroot=login-large_cle_6.0.UP02-build6.0.2042_sles_12-created20161215
ids=10.128.0.33 config_set=p0
```

How to interpret the output:

- If the boot image assigned starts with `initrd-compute-large` or `initrd-login-large`, then one of the Cray recipes for Netroot has been used. But this is not a certain indicator of Netroot use, because a site could change the recipe or rename the boot image.
- If one of the kernel parameters assigned to the node starts with `netroot=compute-large` or `netroot=login-large`, then one of the Cray recipes for Netroot has been used for the image root. But again, this is not a certain indicator of Netroot use, because a site could change the recipe or rename the image root.
- If the boot image assigned starts with `initrd-compute-large` or `initrd-login-large` **and** one of the kernel parameters assigned to the node starts with `netroot=compute-large` or `netroot=login-large`,

that means that Netroot is being used. In this case, there should be a strong similarity between the name of the boot image and the value assigned to the Netroot kernel parameter.

- If there is no kernel parameter `netroot=` at all, then the node is using the tmpfs method, not Netroot. A node using tmpfs should not have a boot image assigned such as `initrd-compute-large` or `initrd-login-large` because the `initrd-*` boot images need a matching image root, which would be specified in the Netroot kernel parameter.

The HSN boot of a tmpfs image is described in *Booting tmpfs Method with bnd* on page 48.

The HSN boot of a Netroot image is described in *Booting Netroot Method with bnd* on page 49.

About the logs generated by cray-ansible and Ansible

The log files that are generated by cray-ansible and the Ansible plays it calls are described in *cray-ansible and Ansible Logs on a CLE Node* on page 51.

Booting with PXE Boot for Boot and SDB Nodes

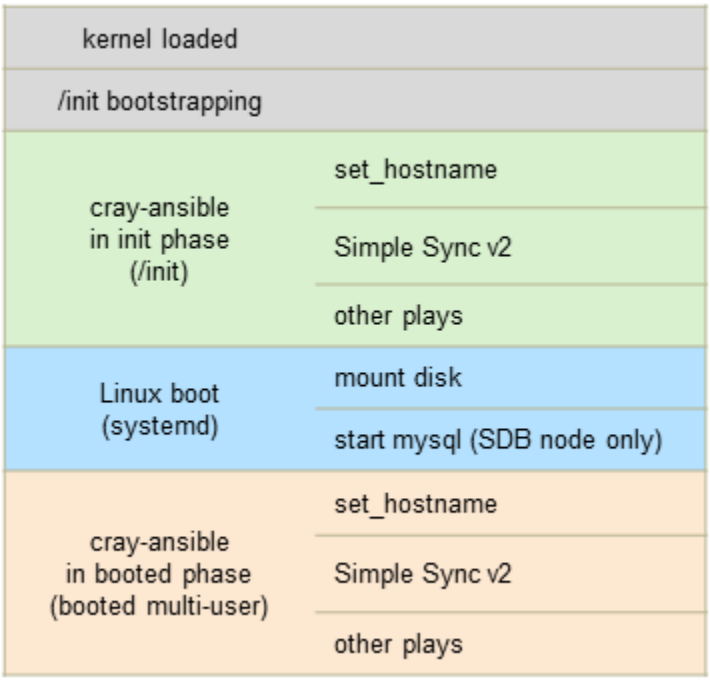
The boot and SDB nodes have an Ethernet connection to the SMW and are capable of being booted over that Ethernet connection using a PXE boot. The Preboot Execution Environment (PXE) uses Dynamic Host Configuration Protocol (DHCP) and Trivial File Transfer Protocol (TFTP) as well as code in the node BIOS to boot the node from software received over the network.

> **NOTE:** There is a limit on the size of boot image for PXE boot. If this limit is exceeded, then `xtbootsys` will display the message "Initramfs too big for PXE boot." This size can be adjusted in `/opt/cray/hss/default/etc/bm.ini`, but there is a maximum size that cannot be exceeded.

The cray-ansible program, a wrapper for Ansible plays, discovers Ansible plays, orders them, and gathers system information. It executes twice during the process of booting with a PXE boot. The first execution occurs when `/init` calls cray-ansible before Linux starts up; this is called the *init* phase, as shown in this figure. During the init phase, Ansible plays create config files for services in preparation for launching those services later in the process, prepare system storage (LVM volume groups, volumes, and file systems), and set up HSN network interfaces. After the standard Linux startup, where systemd boots the system and mounts file systems, cray-ansible runs a second time; this is called the *booted* phase. During the booted phase, Ansible completes the configuration of all services and launches them.

The tasks are run in the order shown in this figure for both the boot and SDB nodes, but starting the mysql database applies to SDB nodes only.

Figure 7. Node Image at Boot Time using PXE Boot



In both the init and booted phases, cray-ansible orders Ansible plays through *directives* included in the plays: run_early, run_late, run_after (specifies dependencies), and run_before (also specifies dependencies, reserved for site use only). Dependencies take precedence over the other two directives, and plays without directives are run somewhere between early and late. Regardless of directives, all plays are run during each phase (though some may be no ops in one of the phases). For example, a play with the run_early directive will be run early in the init phase and early in the booted phase.

Here is the sequence of events for a PXE boot.

1. `xtbounce` triggers the node power on, which will cause the node to run node BIOS.
2. Successful completion of node BIOS leaves the message "Wait4boot" on the console.
3. When `xtbootsys` calls `xtcli boot` for the node, then the node begins the PXE boot process
4. The PXE boot process has the node request an IP address from the SMW via DHCP, then transfer the boot image via the TFTP over the SMW's eth3 to the node's eth0 network connection.
5. `/init` from the boot image executes next. There are many actions done in this script: read kernel parameters from `/proc/cmdline`, initialize logging, load kernel modules, probe for devices, load RCA, make the global config set and CLE config set available for cray-ansible.
6. `/init` calls cray-ansible in the init phase. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-init
   /var/opt/cray/log/ansible/file-changelog-init
   /var/opt/cray/log/ansible/file-changelog-init.yaml
   ```

7. `/init` finishes and transfers control to `systemd`.
8. `systemd` mounts file systems from `/etc/fstab`, starts all enabled services, and so forth.

9. cray-ansible runs in the booted phase. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

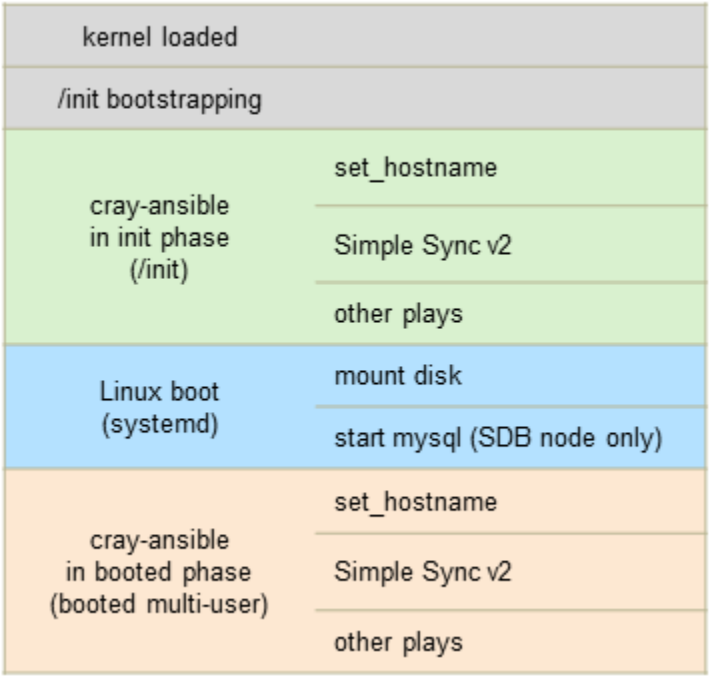
```
/var/opt/cray/log/ansible/sitelog-booted
/var/opt/cray/log/ansible/file-changelog-booted
/var/opt/cray/log/ansible/file-changelog-booted.yaml
```

When boot node failover or SDB node failover is configured, both the primary node and the alternate node must be set in the HSS database and in the CLE config set. Both of the boot nodes (or both of the SDB nodes) will boot at the same time, but there is a pause in `/init`, before calling cray-ansible, if the node detects from a kernel parameter that it is the alternate node. The alternate node will wait in that state until it receives an RCA event that the primary node has failed, at which point it will continue running in `/init` and complete the rest of the booting steps.

Booting tmpfs Method with bnd

As with the PXE boot of the boot and SDB nodes, cray-ansible executes twice during the process of booting a node over the high speed network using a tmpfs image. The tasks are run in the order shown in this figure, but if there are no disks to mount by systemd, then none will be mounted. Likewise, if no mysql database should be running on this node, it will not be started.

Figure 8. Node Image at Boot Time using tmpfs



Here is the sequence of events for a high speed network (HSN) boot of a tmpfs image with the boot node daemon (`bnd`):

1. `xtbounce` triggers the node power on, which will cause the node to run node BIOS.
2. Successful completion of node BIOS leaves the message "Wait4boot" on the console.
3. When `xtbootsys` calls `xtcli boot` for the node, then bnd on the boot node will extract files from the boot image to transfer to the node's memory.

4. `/init` from the boot image executes next. There are many actions done in this script: read kernel parameters from `/proc/cmdline`, initialize logging, load kernel modules, probe for devices, load RCA, and make the global config set and CLE config set available for cray-ansible.
5. `/init` calls cray-ansible in the init phase. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-init
   /var/opt/cray/log/ansible/file-changelog-init
   /var/opt/cray/log/ansible/file-changelog-init.yaml
   ```

6. `/init` finishes and transfers control to `systemd`.
7. `systemd` mounts file systems from `/etc/fstab`, starts all enabled services, and so forth.
8. cray-ansible runs in the booted phase. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-booted
   /var/opt/cray/log/ansible/file-changelog-booted
   /var/opt/cray/log/ansible/file-changelog-booted.yaml
   ```

When the console for a node is in the DEBUG shell, connect with the `xtcon` command from the SMW to that node.

```
smw# xtcon c0-0c0s0n1
```

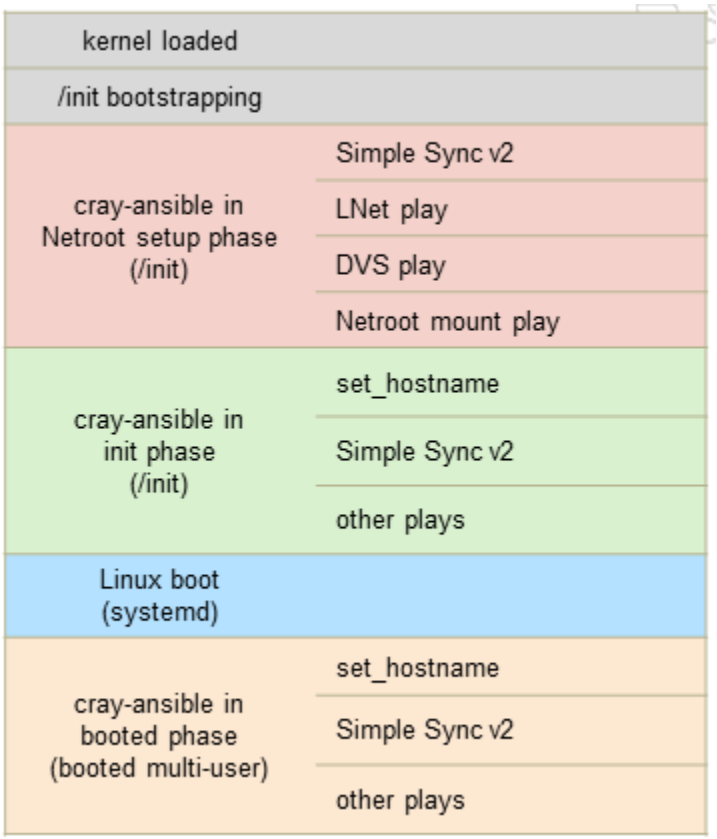
NOTE: To disconnect an `xtcon` session from the console of a node, type **`^]`** to quit.

If the DEBUG=true kernel parameter is set, then there are several breakpoints at which `/init` will drop into the DEBUG shell. See the `/init` script in the boot image or on the image root in `/var/opt/opt/cray/imps/image_roots/$IMAGE` on the SMW to identify these breakpoints.

Booting Netroot Method with bnd

The cray-ansible program executes three times during the process of booting a node over the high speed network using a Netroot image, as compared to only twice when booting a tmpfs image. The additional phase, called the *Netroot setup* phase, occurs just before the init phase, as shown in this figure. During the Netroot setup phase, only Ansible plays of type *netroot_setup* are run, and the netroot image is mounted so that it can be accessed later in the boot process.

Figure 9. Node Image at Boot Time using Netroot



Here is the sequence of events for an HSN boot of a Netroot image with the boot node daemon (`bnd`):

1. `xtbounce` triggers the node power on, which will cause the node to run node BIOS.
2. Successful completion of node BIOS leaves the message "Wait4boot" on the console.
3. When `xtbootsys` calls `xtcli boot` for the node, then `bnd` on the boot node will extract files from the boot image to transfer to the node's memory.
4. `/init` from the boot image executes next. There are many actions done in this script: read kernel parameters from `/proc/cmdline`, initialize logging, load kernel modules, probe for devices, load RCA, make the global config set and CLE config set available for cray-ansible.
5. `/init` calls cray-ansible in the Netroot setup phase, in which cray-ansible runs only plays of type netroot-setup. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, `cray-ansible` will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-init-netroot_setup
   /var/opt/cray/log/ansible/file-changelog-init-netroot_setup
   /var/opt/cray/log/ansible/file-changelog-init-netroot_setup.yaml
   ```

6. `/init` calls cray-ansible in the init phase, in which cray-ansible runs only plays of type cle from the Netroot boot image. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-init
   /var/opt/cray/log/ansible/file-changelog-init
   ```

```
/var/opt/cray/log/ansible/file-changelog-init.yaml
```

7. `/init` finishes and transfers control to `systemd`.
8. `systemd` mounts file systems from `/etc/fstab`, starts all enabled services, and so forth.
9. cray-ansible runs in the booted phase, in which cray-ansible runs only plays of type cle from the Netroot image root. If this fails, then the node will drop into the DEBUG shell. If it succeeds, then `/init` continues. Regardless, cray-ansible will log to:

   ```
   /var/opt/cray/log/ansible/sitelog-booted
   /var/opt/cray/log/ansible/file-changelog-booted
   /var/opt/cray/log/ansible/file-changelog-booted.yaml
   ```

cray-ansible and Ansible Logs on a CLE Node

Because `cray-ansible` is run more than once during the boot of a node, there are different log files to inspect for issues. cray-ansible is called first by `/init` and later by `systemd` after transitioning into multi-user mode. For nodes with tmpfs images, there is a single call to cray-ansible by `/init`. For nodes with Netroot images, the first call to cray-ansible by `/init` is to do Netroot setup using the Ansible plays in the initrd (boot image), which will mount the Netroot image root via tier2 nodes from the boot node. The second call to cray-ansible by `/init` uses the Ansible plays in the Netroot image root. If cray-ansible fails during the init phase, it will display this message on the console:

```
cray-ansible: /etc/ansible/site.yaml completed in init - FAILED
```

And immediately above that line will be the Ansible play recap and before that the last Ansible task executed that had an error causing cray-ansible to fail. There may be more context in the full Ansible logs on the node in the `/var/opt/cray/log/ansible` directory, but this will provide the first hint as to what failed.

```
2016-03-24T07:28:05.382423-05:00 c0-0c0s2n0 PLAY RECAP
*********************************************
2016-03-24T07:28:05.382437-05:00 c0-0c0s2n0            to retry, use: --limit @/
root/site.yaml.retry
2016-03-24T07:28:05.382451-05:00 c0-0c0s2n0
2016-03-24T07:28:05.382465-05:00 c0-0c0s2n0 localhost       : ok=109  changed=14
unreachable=0    failed=1
2016-03-24T07:28:05.382480-05:00 c0-0c0s2n0
2016-03-24T07:28:05.382495-05:00 c0-0c0s2n0 Failed Ansible configuration
```

After the init phase of cray-ansible is done, `/init` will transfer control to `systemd` to start multi-user mode. In multi-user mode, cray-ansible will be called again in the booted phase. Logs on the node are in `/var/opt/cray/log/ansible`.

Netroot setup phase	`sitelog-init-netroot_setup`
	`file-changelog-init-netroot_setup`
	`file-changelog-init-netroot_setup.yaml`

init phase	`sitelog-init` has Ansible play output. `file-changelog-init` shows each file changed by an Ansible play. `file-changelog-init.yaml` shows each file changed by an Ansible play in YAML.
booted phase	`sitelog-booted` has Ansible play output. `file-changelog-booted` shows each file changed by an Ansible play.

`file-changelog-booted.yaml` shows each file changed by an Ansible play in YAML.

A Cray-specific Ansible plugin will track to the `file-changelog-*` files all files changed by Ansible modules affecting files: acl, assemble, blockinfile, copy, fetch, file, find, ini_file, lineinfile, patch, replace, stat, synchronize, template, unarchive, xtattr.

The `sitelog-*` files show output from each task in executed plays.

```
2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
****************************************
2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true, "cmd":
"grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print $2}'", "delta":
"0:00:00.002536",
"end": "2016-01-17 12:15:27.471384", "rc": 0, "start": "2016-01-17
12:15:27.468848", "stderr": "",
"stdout": "6.0.UP01", "warnings": []}
```

The location of a task can be found in the Ansible plays. Since the TASK above had the name "task motd, release," search for that in the locations that have Ansible plays. In `/etc/ansible` are the plays from the software image root, and the two locations under `/etc/opt/cray/config` are the current CLE config set and the global config set, which might have site-local plays.

```
boot# grep -Rn "task motd, release" /etc/ansible \
/etc/opt/cray/config/current/ansible /etc/opt/cray/config/global/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

The `file-changelog-*` logs show the timestamp, the Ansible phase (Netroot setup, init, booted), the Ansible module, the file that was changed, which play changed it, and information about owner, group, and permission mode on the file.

- `file-changelog-init`:

  ```
  Apr 05 2016 21:07:47 (init) template: file '/etc/nologin' changed by Ansible
  task file
  '/etc/ansible/roles/early/tasks/nologin.yaml' with owner=root, group=root,
  mode=0775
  ```

- `file-changelog-booted`:

  ```
  Apr 05 2016 16:09:43 (booted) lineinfile: file '/etc/sysconfig/nfs' changed by
  Ansible
  task file '/etc/ansible/roles/fs_share/tasks/nfs_shares.yaml' with owner=None,
  group=None, mode=None
  ```

Commands Helpful in Troubleshooting a Boot

This section describes commands that may be helpful in troubleshooting a boot. The commands and output provided here may be referenced from specific scenarios in the next section, *Techniques for Troubleshooting a Failed Boot* on page 69.

Check RSMS Daemons

There are two ways to check whether the daemons started by rsms are running.

- Method 1: Use the `systemctl` command to query systemd.

```
smw# systemctl status -l -n 99 rsms
rsms.service - hss daemon control
   Loaded: loaded (/usr/lib/systemd/system/rsms.service; enabled)
   Active: active (exited) since Wed 2016-07-06 22:07:19 CDT; 12h ago
   Process: 6744 ExecStart=/opt/cray/hss/default/bin/hssctl start (code=exited,
status=0/SUCCESS)

Jul 06 22:07:19 smw hssctl[6744]: Starting daemons: erd erdh state_manager
nid_mgr bootmanager sedc_manager xtpmd erfsd xtremoted xtpowerd nimsd xtsnmpd
xtdiagd cabroutes boot_cmds sec_cmd
```

- Method 2: Run the `/etc/inti.d/rsms` script with the status option.

```
smw# /etc/init.d/rsms status
PID             DAEMON             STATE          UPTIME
53430           erd                running        Tue 2016-06-21 22:34:54 CDT
53732           erdh               running        Tue 2016-06-21 22:34:57 CDT
53927           state_manager      running        Tue 2016-06-21 22:34:58 CDT
54062           nid_mgr            running        Tue 2016-06-21 22:34:59 CDT
54196           bootmanager        running        Tue 2016-06-21 22:35:00 CDT
54330           sedc_manager       running        Tue 2016-06-21 22:35:01 CDT
55253           xtpmd              running        Tue 2016-06-21 22:35:10 CDT
56092           erfsd              running        Tue 2016-06-21 22:35:14 CDT
56510           xtremoted          running        Tue 2016-06-21 22:35:19 CDT
57482           xtpowerd           running        Tue 2016-06-21 22:35:23 CDT
57910           nimsd              running        Tue 2016-06-21 22:35:27 CDT
58107           xtsnmpd            running        Tue 2016-06-21 22:35:31 CDT
58876           xtdiagd            running        Tue 2016-06-21 22:35:35 CDT
```

Check diod daemon

To check whether the distributed I/O daemon (diod) is running on the SMW:

```
smw# systemctl status -l -n 99 cray-ids-service
cray-ids-service.service - cray-ids-service server
   Loaded: loaded (/usr/lib/systemd/system/cray-ids-service.service; disabled)
   Active: active (running) since Tue 2016-06-21 22:34:52 CDT; 3 weeks 0 days ago
```

```
 Main PID: 53484 (diod)
   CGroup: /system.slice/cray-ids-service.service
           +-53484 /usr/sbin/diod -U root --export-opts ro --allsquash --no-a...
```

When the output from the `systemctl status` command shows a line with "active (running)," it means the daemon being checked is running.

```
Active: active (running) since Tue 2016-06-21 22:34:52 CDT; 3 weeks 0 days ago
```

Check cray-cfgset-cache Daemon

To check whether the cfgset-cache daemon is running on the SMW:

```
smw# systemctl status -l -n 99 cray-cfgset-cache
cray-cfgset-cache.service - Automatic cache generation for config sets
   Loaded: loaded (/usr/lib/systemd/system/cray-cfgset-cache.service; disabled)
   Active: active (running) since Tue 2016-06-21 22:35:58 CDT; 2 weeks 5 days ago
 Main PID: 62972 (python)
   CGroup: /system.slice/cray-cfgset-cache.service
           тт62972 python /opt/cray/imps-distribution/default/bin/cfgset-cache

Jul 11 14:23:33 smw cfgset-cache[62972]: IDS INFO - Backgrounded cfgset 'p0'
squashfs cache generation.
Jul 11 14:23:37 smw cfgset-cache[62972]: IDS INFO - 361 changes to p0 noted during
last grace window.
Jul 11 14:23:41 smw cfgset-cache[62972]: IDS INFO - 134 changes to p0 noted during
last grace window.
Jul 11 14:23:45 smw cfgset-cache[62972]: IDS INFO - 14 changes to p0 noted during
last grace window.
Jul 11 14:23:48 smw cfgset-cache[62972]: IDS INFO - Backgrounded cfgset 'p0-
autosave-2016-07-11T14:23:48' squashfs cache generation.
Jul 11 14:23:49 smw cfgset-cache[62972]: IDS INFO - 46 changes to p0 noted during
last grace window.
Jul 11 14:23:53 smw cfgset-cache[62972]: IDS INFO - Aggregated 4 changes to cfgset
'p0-autosave-2016-07-11T14:23:48' cache over 4.1 second window.
Jul 11 14:23:53 smw cfgset-cache[62972]: IDS INFO - Aggregated 557 changes to
cfgset 'p0' cache over 20.1 second window.
Jul 11 14:24:29 smw cfgset-cache[62972]: IDS INFO - Backgrounded cfgset 'p0'
squashfs cache generation.
Jul 11 14:24:34 smw cfgset-cache[62972]: IDS INFO - Aggregated 91 changes to
cfgset 'p0' cache over 4.1 second window.
```

Check DHCP or TFTP Daemons

To check whether the Dynamic Host Configuration Protocol (DHCP) server daemon (dhcpd) is running on the SMW:

```
smw# systemctl status -l -n 99 dhcpd
```

To check whether the Trivial File Transfer Protocol (TFTP) server daemon (atftpd) is running on the SMW:

```
smw# systemctl status -l -n 99 atftpd
```

Check Console Messages

Use the `xtconsole` command to check console messages. This command subscribes to ec_console_log events from erd and logs them to stdout. When `xtconsole` is started during a boot session, stdout is captured and redirected to a log file in the boot session directory.

One advantage of using `xtconsole` interactively (i.e., watching stdout) over looking at the log in `/var/opt/cray/log/p0-current/console.*` is that this command can be running across multiple boot sessions. The advantage of the log for a boot session is that it can be analyzed after a problem has occurred.

This example displays console messages for all nodes and adds a timestamp at the beginning of each line.

```
smw# xtconsole -at
```

This example displays the console messages, with timestamps, for a particular node (c0-0c0s0n1 in the example).

```
smw# xtconsole -t c0-0c0s0n1
```

Log In to a Node

healthy system

When a node is operating normally, it should be possible to use `ssh` to log in to the node. If the node has partially booted, then it might have booted far enough to start the sshd daemon before hitting a failure.

from SMW

From the SMW, it should be possible to log in to the boot and SDB nodes directly.

```
smw# ssh boot
smw# ssh sdb
```

from boot or SDB node

From the boot or SDB nodes, it should be possible to log in to any other node.

```
boot# ssh c0-1c2s7n2
sdb# ssh c1-2c0s15n3
```

from outside

From anywhere outside the XC system, it should be possible to log in to the login node or gateway node or other nodes with external network connections, unless firewall rules prevent it. Typically, `ssh` is permitted through any firewall.

```
user@host> ssh mycray-login
```

unhealthy system

If unable to log in using `ssh`, then use `xtcon` to connect. `xtcon` is a two-way (input and output) console program used mainly during hardware bring-up and debugging to connect to XC nodes. See the man page for more details.

```
smw# xtcon c0-1c2s7n2
```

```
user@host> ssh mycray-smw
smw# xtcon c0-1c2s7n2
```

Check Daemons Using xtalive

The `xtalive` program sends an event to daemons on the SMW, cabinet controllers (CC), or blade controllers (BC) and waits for the appropriate number of events in response. It is basically a 'ping' to specific daemons to check whether they are running and responding to HSS events.

```
crayadm@smw> xtalive
crayadm@smw> xtalive -a DAEMON
```

In the second command, substitute for *`DAEMON`* one of these daemons that can be targeted by `xtalive`.

Table 2. Targetable Daemons and Where They are Located

daemon	located on
erd	SMW, CC, BC
state_manager	SMW
bootmanager	SMW
bnd	boot node daemon: boot node
bcsysd	BC
ccsysd	CC
bcrtrd	BC
nid_mgr	SMW
bcnwd	BC
l0dd	BC
bcbwtd	BC
xtnlrd	SMW
ccrd	CC
ccrdhelper	BC
ITP	BC

Check STONITH on Blade Controller

There are many configuration settings that could be changed on cabinet controller (CC) or blade controller (BC) daemons with `xtdaemonconfig`, but the one of interest here is whether STONITH has been enabled on the blade. Normally STONITH is disabled for all blades. However, when a system has been configured for boot and/or SDB failover, STONITH is enabled on the blade containing the primary boot node and the primary SDB node to ensure that proper failover from primary node to alternate node will happen when the primary node misses a heartbeat.

For CLE releases 6.0.UP00 and UP01, a problem occurs when STONITH is set on a blade, and any node on that blade drops into the DEBUG shell while booting. While in the DEBUG shell, the node will fail to heartbeat and will be halted with an NMI (non-maskable interrupt). It does not matter whether the node dropped into the DEBUG

shell due to an error from cray-ansible or whether the kernel parameter DEBUG=true is set. This problem was fixed in CLE 6.0.UP02.

Use this command to check the STONITH setting.

```
crayadm@smw> xtdaemonconfig | grep stonith=true
```

Check for Cabling Issues

The `xtcablecheck` utility compares the link endpoint pairs as known to the routing software with the link ID values set in MMRs in each link control block (LCB) in the Aries ASIC. When links are deadstarted, they exchange their physical ID with their peer at the other end of the link. `xtcablecheck` verifies that the cabling is correct by comparing the IDs that were exchanged with what is expected, given the configuration of the system.

If `xtbootsys` fails due to an error from `xtcablecheck`, then fix the issue and confirm that `xtcablecheck` has no error.

```
crayadm@smw> xtbounce p0
```

```
crayadm@smw> xtcablecheck p0
```

Check Hardware Inventory

The hardware inventory can be checked with `xthwinv`, which requests blade and node attributes from all blades, or specified blades, in the system. The output can be human readable, or in XML format for parsing. The XML version is used by `xtbootsys` to pass information on nodes to the boot node, where a Perl script parses the information and stores it into the SDB database when the boot node comes up.

> **NOTE:** This command requires that the state_manager be running on the SMW and that a successful `xtbounce` has completed.

```
crayadm@smw> xthwinv p0
```

Check Boot Configuration

The `xtcli part_cfg` command shows information about each partition defined in the system. The entire system is defined to be p0, but other partitions could be p1 through p31.

Is the partition enabled? If not enabled, then it cannot be booted. Does this output show the expected members (components) in the partition? Are the correct nodes listed for boot and SDB? Are the correct boot images listed?

```
crayadm@smw> xtcli part_cfg show
Network topology: class 0
=== part_cfg ===
------------------
[partition]: p0: enable (noflags|)
[members]: c0-0
[boot]: c0-0c0s0n1:halt,c0-0c1s0n1:halt
[sdb]: c0-0c0s1n1:halt,c0-0c1s1n1:halt
[NIMS_image 0]: /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
[NIMS_image 1]: /var/opt/cray/imps/boot_images/service_cle_6.0.UP01-
```

```
build6.0.96_sles_12-created20160614.cpio
[NIMS_image 2]: /var/opt/cray/imps/boot_images/dal_cle_6.0.UP01-
build6.0.96_centos_6.5-created20160614.cpio
[NIMS_image 3]: /var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
[NIMS_image 4]: /var/opt/cray/imps/boot_images/fio-service_cle_6.0.UP01-
build6.0.96_sles_12-created20160615.cpio
```

NOTE: This command requires that the state_manager and nimsd be running.

Enable or Disable a Component

If a boot has an issue with a few of the nodes of a certain type or function, it may be necessary to disable that component for later hardware action. If a node fails during `xtbounce` repeatedly, it may be expedient to disable that node so that `xtbounce` will succeed.

```
crayadm@smw> xtcli disable c0-0c0s11n2
```

And once the hardware issue has been resolved, enable the node again.

```
crayadm@smw> xtcli enable c0-0c0s11n2
```

If large numbers of blades are disabled, this may cause a routing problem on the HSN, depending on which slots and chassis have the disabled components.

Check Status of Nodes

There are many ways to check the status of nodes.

from SMW

From the SMW, any hardware component in the HSS database can be checked.

```
crayadm@smw> xtcli status p0
crayadm@smw> xtcli status c1-2
crayadm@smw> xtcli status c0-0c0s15n0
```

from boot node

```
user@boot> xtprocadmin
```

from login node

```
user@login> xtprocadmin
```

```
user@login> xtnodestat
```

```
user@login> apstat
```

Change Node Role Between Service and Compute

The configuration of CLE may require that some compute nodes be repurposed as service nodes. These are called repurposed compute nodes (RCN) and are commonly used to provide tier2 nodes for Cray scalable services. If a node is in the CLE config set as a tier2 node, then it must be marked as a service node in the HSS database and must boot a service node boot image.

These commands set or check the node type in the HSS database.

set as service

```
smw# xtcli mark_node service c0-0c0s15n0
```

set as compute

```
smw# xtcli mark_node compute c0-0c0s15n0
```

check node type

```
smw# xtcli status c0-0c0s15n0
```

Check NIMS Map

Use the `cmap` command to show which NIMS map is the active map for a partition. This example shows that the currently active map is the p0 map. The others were created during a fresh installation or were saved as backups when applying patches to the system.

```
smw# cmap list
NAME                                        PARTITION  ACTIVE_MAP
autogenerated_map-0.p0                      p0         False
backup-pre-CLE_6.0.UP01.PS02-1606141402     p0         False
backup-pre-SMW_8.0.UP01.PS02-1606141432     p0         False
p0                                          p0         True
```

Check Which Boot Images Have Been Assigned

Show the boot configuration used by the boot manager. This will list all of the boot images assigned via NIMS.

NOTE: This command requires that the state_manager and nimsd be running.

```
crayadm@smw> xtcli part_cfg show
Network topology: class 0
=== part_cfg ===
------------------
[partition]: p0: enable (noflags|)
[members]: c0-0
[boot]: c0-0c0s0n1:halt,c0-0c1s0n1:halt
[sdb]: c0-0c0s1n1:halt,c0-0c1s1n1:halt
[NIMS_image 0]: /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 1]: /var/opt/cray/imps/boot_images/service_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 2]: /var/opt/cray/imps/boot_images/dal_cle_6.0.UP02-
build6.0.2042_centos_6.5-created20161215.cpio
[NIMS_image 3]: /var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 4]: /var/opt/cray/imps/boot_images/fio-service_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
```

Check Node NIMS Group, Boot Image, and Kernel Parameter Assignment

Check the information assigned to nodes for the currently active NIMS map. For each node the NIMS group, boot image, and kernel parameters which have been assigned are displayed.

This example shows only three nodes instead of the entire set of nodes in the XC system. The boot node is c0-0c0s0n1, which boots from a service boot image. The compute node boots with a Netroot boot image (initrd-compute-large) and `netroot=compute-large` kernel parameter. The login node boots with a tmpfs boot image.

```
smw# cnode list
 Node          Type
Group                                                                  Image
                                                   Loadfile    Config Set   Parameters
c0-0c0s0n1   service  service
  /var/opt/cray/imps/boot_images/service_cle_6.0.UP01-build6.0.68_sles_12-
created20160212.cpio
                -           NIMS_GROUP=service ids=10.128.0.130,10.128.0.138
config_set=p0
c0-0c0s10n3  compute  compute
  /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.68_sles_12-created20160212.cpio
  -           NIMS_GROUP=compute netroot=compute-large_cle_6.0.UP01-
build6.0.68_sles_12-created20160210
 ids=10.128.0.130,10.128.0.138 config_set=p0
c0-0c1s1n1   service  login
    /var/opt/cray/imps/boot_images/login_cle_6.0.UP01-build6.0.68_sles_12-
created20160212.cpio
                 -           NIMS_GROUP=login ids=10.128.0.130,10.128.0.138
config_set=p0
```

To show only certain components, use a space-separated component list or a list with wildcards.

```
smw# cnode list c0-0c0s0n1 c0-0c1s0n1 c0-0c0s1n1 c0-0c1s1n1
smw# cnode list c0-0c0s1*
```

Check Whether Node is Using Netroot or tmpfs

To check whether node c0-0c0s0n2 is using Netroot from the SMW, run this command:

```
smw# cnode list c0-0c2s0n1
NAME         TYPE     GROUP
IMAGE
                                          CONFIG_SET  EXT_PARAMETERS
c0-0c0s0n2   service  login
/var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
    -           sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 NIMS_GROUP=login
netroot=login-large_cle_6.0.UP02-build6.0.2042_sles_12-created20161215
ids=10.128.0.33 config_set=p0
```

How to interpret the output:

- If the boot image assigned starts with `initrd-compute-large` or `initrd-login-large`, then one of the Cray recipes for Netroot has been used. But this is not a certain indicator of Netroot use, because a site could change the recipe or rename the boot image.
- If one of the kernel parameters assigned to the node starts with `netroot=compute-large` or `netroot=login-large`, then one of the Cray recipes for Netroot has been used for the image root. But again, this is not a certain indicator of Netroot use, because a site could change the recipe or rename the image root.

- If the boot image assigned starts with `initrd-compute-large` or `initrd-login-large` **and** one of the kernel parameters assigned to the node starts with `netroot=compute-large` or `netroot=login-large`, that means that Netroot is being used. In this case, there should be a strong similarity between the name of the boot image and the value assigned to the Netroot kernel parameter.
- If there is no kernel parameter `netroot=` at all, then the node is using the tmpfs method, not Netroot. A node using tmpfs should not have a boot image assigned such as `initrd-compute-large` or `initrd-login-large` because the `initrd-*` boot images need a matching image root, which would be specified in the Netroot kernel parameter.

Check Which Boot Images Exist on the System

After using `xtcli part_cfg show` to see which boot images are needed for booting or `cnode list` to see which boot images are assigned to which nodes, check that those exact boot image names have been created.

```
smw# ls -l /var/opt/cray/imps/boot_images

smw# ls -l /var/opt/cray/imps/boot_images/mybootimagename
```

Check Which Image Roots Exist on the System

Check which images have been built from recipes into image roots.

List all images that have been built from recipes.

```
smw# image list
compute-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615
fio-service_cle_6.0.UP01-build6.0.96_sles_12-created20160615
initrd-compute-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615
initrd-login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615
login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615
service_cle_6.0.UP01-build6.0.96_sles_12-created20160614
```

Show a specific image to see what path its image root has.

```
smw# image show imageroot
compute-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615:
  name: compute-large_cle_6.0.UP01-build6.0.96_sles_12-created20160615
  created: 2016-06-15T14:55:15
  history:
    2016-06-15T14:55:22: Successful build of Recipe
'seed_common_6.0up01_sles_12_x86-64' into Image 'compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615'.
    2016-06-15T15:05:25: Successful build of top level recipe 'compute-
large_cle_6.0up01_sles_12_x86-64_ari'.
    2016-06-15T15:05:25: Successful rebuild of RPM database.
    2016-06-15T15:50:27: Remotely cloned to host 'boot'.
  path: /var/opt/cray/imps/image_roots/compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160615
```

Confirm that those image roots exist on the SMW and boot node for any Netroot images for compute or login nodes.

```
smw# ls -l /var/opt/cray/imps/image_roots
smw# ls -l /var/opt/cray/imps/image_roots/imageroot
```

```
boot# ls -l /var/opt/cray/imps/image_roots
boot# ls -l /var/opt/cray/imps/image_roots/imageroot
```

Observe Network Traffic on SMW Network Interfaces

It may be necessary to check for network traffic on the SMW using `tcpdump` or `wireshark`. The `wireshark` program requires an X display server.

```
smw# tcpdump
```

```
smw# wireshark
```

Table 3. What to Monitor in Certain Situations

What to Monitor	Situation
eth0	If a problem is suspected with the management network (external to the SMW).
eth1	If a problem is suspected with the HSS network (to the CCs).
eth3	If a problem is suspected with the admin network (to the boot and SDB nodes).
eth2 and eth4	If SMW HA is being used, and a problem is suspected with heartbeat between the new SMWs.
eth5	If SMW HA is being used, and a problem is suspected with DRBD (distributed replicated block device) not keeping the Postgresql database (used for power management) synchronized between the two SMWs.

Check Firewall

The firewall is normally enabled on the SMW with eth0 in the FW_DEV_EXT (external) zone and all other interfaces on the SMW in the FW_DEV_INT (internal) zone.

Are the firewall daemons running?

```
smw# systemctl status -l -n 99 SuSEfirewall2_init
smw# systemctl status -l -n 99 SuSEfirewall2
```

What are the iptables rules?

```
smw# iptables -L
```

Search a Config Set

Use Search to Locate Settings in a Config Set

The search subcommand is helpful when a user wants to view or change a configuration parameter (setting) but does not know which configuration template or worksheet contains it. To search for a configuration setting/field name or value, use the `cfgset search` command:

```
smw# cfgset search --term myvalue CONFIGSET
```

Search tips:

- To broaden a search, use multiple search terms (a logical OR).
- To narrow a search, use state and level filters.
- Unlike the `create` and `update` subcommands, the `search` subcommand has a default value of `all` for the state filter.

Here's an example that searches for the terms `c0-0c0s1n1` and `lus/` in settings of any level in config set p0:

```
smw# cfgset search --term c0-0c0s1n1 --term lus/ --level advanced p0
```

The configurator outputs highlighted dotted-path notation matches to the search term in a per-service report:

```
# 1 match for 'c0-0c0s1n1' from cray_scalable_services_config.yaml
#-------------------------------------------------------------------------------
cray_scalable_services_data.settings.scalable_service.data.tier1: c0-0c0s0n1, c0-0c0s1n1
# 1 match for 'lus/' from cray_node_health_config.yaml
#-------------------------------------------------------------------------------
cray_node_health_.settings.filesys_plugins.data.Default Filesystem.path: /lus/case1
...(more matches not included in example)
```

To output more information about the fields and values that match the search term(s), add the `--format full` command line option. This will display meta information about the setting in which the term was found, such as its level, state, and default value.

Note that the search subcommand does not search guidance text in the configuration templates and worksheets.

Use Search to Print out the Entire Config Set

To print out an entire config set, simply search the config set and omit the `--term` option. For example, to view all required fields that have not been set in config set p0, use the following command:

```
smw# cfgset search --level required --state unset p0
```

List the Ansible Playbooks in a Config Set and Image Root

The `ansible_cfg_search` command on the SMW can list the Ansible plays in a certain config set and image root.

```
smw# module load system-config
smw# ansible_cfg_search -q p0 custom_compute_cle
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/allow_users.yaml
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/alps.yaml
...
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/set_hostname.yaml
...
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/sysenv.yaml
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/wlm_detect.yaml
```

Search the Ansible Playbooks in a Config Set and Image Root

The `ansible_cfg_search` command on the SMW will search Ansible plays in a certain config set and image root to see which plays use which config set data items.

This example shows the specific files included by the set_hostname.yaml playbook which use data items from the cray_net config service.

```
smw# ansible_cfg_search -p set_hostname.yaml p0 custom_compute_cle
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/set_hostname.yaml:
  - /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/roles/
set_hostname/tasks/main.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_netroot_preload_config.yaml:
      - cray_net.settings.hosts.data
    - /var/opt/cray/imps/config/sets/global/config/
cray_network_boot_packages_config.yaml:
      - cray_net.settings.hosts.data
...
  - /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/
set_hostname.yaml:
    - /var/opt/cray/imps/config/sets/global/config/
cray_network_boot_packages_config.yaml:
      - cray_net.enabled
      - cray_net.settings.service.data.cray_managed
...
```

This example searches for the specified setting (`cray_alps.settings.common.data.xthostname`) in the config set files and Ansible files in the image root to determine which Ansible plays do something with its data.

```
smw# module load system-config
smw# ansible_cfg_search p0 service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614 \
-s cray_alps.settings.common.data.xthostname
/var/opt/cray/imps/image_roots/service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614/etc/ansible/alps.yaml:
  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614/etc/ansible/roles/alps/tasks/service.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_alps_config.yaml:
      - cray_alps.settings.common.data.xthostname

/var/opt/cray/imps/image_roots/service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614/etc/ansible/compute_node.yaml:
  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614/etc/ansible/roles/compute_node/tasks/xthostname.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_alps_config.yaml:
      - cray_alps.settings.common.data.xthostname
```

Search Ansible Plays on a Node

The grep command can search the three locations on a node where Ansible plays can be stored.

```
node# grep -r "TERM" /etc/ansible /etc/opt/cray/config/current /etc/opt/cray/
config/global
```

One way to use this is to identify the name of an Ansible task that failed in an Ansible log file, and then locate the file containing the Ansible task and read the code to understand what was being attempted.

In this example Ansible log, the task that failed is "task motd, release."

```
2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
****************************************
2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true,
"cmd": "grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print $2}'",
"delta": "0:00:00.002536",
"end": "2016-01-17 12:15:27.471384", "rc": 0, "start": "2016-01-17
12:15:27.468848", "stderr": "",
"stdout": "6.0.UP02", "warnings": []}
```

The location of a task can be found in the Ansible plays. Because the example shows that the failed task has the name "task motd, release," search for that name in the locations that have Ansible plays:

- `/etc/ansible` has Ansible plays from the software image root
- `/etc/opt/cray/config/current` has Ansible plays from the current CLE config set, which may include site-local plays
- `/etc/opt/cray/config/global` has Ansible plays from the global config set, which may include site-local plays

```
boot# grep -Rn "task motd, release" /etc/ansible \
/etc/opt/cray/config/current/ansible /etc/opt/cray/config/global/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

The grep output shows that the file containing this task is `/etc/ansible/roles/cle_motd/tasks/motd.yaml`. Look at line 15 in that file to determine what that chunk of code is doing.

Check for Warnings, Alerts, and Reservations

There are several related `xtshow*` commands that can be used on the SMW to display the status of XC system components. These commands can have a comma-separated or space-separated ID list of components on the command line or the `-f filename` option, where the file name has a list of the nodes upon which to act.

```
xtshow_alert
xtshow_class
xtshow_compute
xtshow_diag
xtshow_disabled
xtshow_empty
xtshow_error
xtshow_halt
xtshow_network
xtshow_noflags
xtshow_not_empty
xtshow_off
xtshow_on
xtshow_ready
xtshow_reserve
xtshow_service
xtshow_standby
xtshow_topology
xtshow_warn
```

There are several related `xtclear*` commands that can clear various flags from all components that have those flags currently set. These commands can either have a comma-separated or space-separated idlist of

components on the command line or the `-f filename` option, where the file name has a list of the nodes upon which to act.

```
xtclear_alert
xtclear_reserve
xtclear_warn
```

Check for Locks

The state manager may have created locks that prevent a boot from completing. Use this command to show all locks.

```
smw# xtcli lock show
```

If there are locks that prevent a boot, they can be removed. See the man page for `xtcli`.

Check for PCIe Link Errors

The xtpcimon daemon monitors the health of PCIe (peripheral component interconnect express) channels and logs PCIe link errors to a file.

The `xtpe` command processes pcimon log files for PCIe link errors.

To decode the xtpcimon log files on the SMW, use one of these commands.

- For a single pcimon log file for the current boot session of p0 on a particular date YYYYMMDD:

  ```
  crayadm@smw> xtpe /var/opt/cray/log/p0-current/pcimon-YYYYMMDD
  ```

- For all pcimon log files for the current boot session of p0. This boot log directory is relative to `/var/opt/cray/log`.

  ```
  crayadm@smw> xtpe -b p0-current
  ```

- For a single pcimon log file for a given boot session of p0.

  ```
  crayadm@smw> xtpe /var/opt/cray/log/p0-20160901t180511/pcimon-YYYYMMDD
  ```

- For all pcimon log files for a given boot session of p0. This boot log directory is relative to `/var/opt/cray/log`.

  ```
  crayadm@smw> xtpe -b p0-20160901t180511
  ```

To display those errors in real time, creating reports whenever a signal is received, use the `-g` option.

```
crayadm@smw> xtpe -g
```

See the `xtpe(8)` man page for more information.

Check for Hardware Errors

The xthwerrlogd daemon listens for hardware error events from the ASIC network chip and writes them to a binary file.

The `xthwerrlog` command analyzes that binary file.

To decode the hardware error log (hwerrlog) on the SMW, use one of these commands.

- For all boot sessions of p0:

```
crayadm@smw> xthwerrlog -P /var/opt/cray/log/p0
```

- For the current boot session of p0:

```
crayadm@smw> xthwerrlog -P /var/opt/cray/log/p0-current
```

- for all logs from p0 for June, 2016:

```
crayadm@smw> xthwerrlog -P /var/opt/cray/log/p0-201606
```

To display those errors in real time, as they occur, no options are needed.

```
crayadm@smw> xthwerrlog
```

See the `xthwerrlog(8)` man page for more information.

Check for LCB and Router Errors

The xtnetwatch daemon monitors the system high-speed network (HSN) faults interconnect for link control block (LCB) and router errors, and it logs them to a file.

The `xtle` command analyzes netwatch log files for HSN errors.

To analyze HSN link errors, use one of these commands.

- For the current boot session of p0:

```
crayadm@smw> xtle -b p0-current
```

- For the current boot session of p0 on a particular date YYYYMMDD:

```
crayadm@smw> xtle /var/opt/cray/log/p0-current/netwatch-YYYYMMDD
```

To display HSN link errors in real time, creating reports whenever a signal is received, use this command.

```
crayadm@smw> xtle -g
```

See the `xtle(8)` man page for more information.

Check Time on a Node

To access the time on a node, use these commands, which check the time without changing it. There is no change to the node.

- To check the real-time clock (RTC) time:

```
node# hwclock
```

- To check system time:

```
node# date
```

Techniques for Troubleshooting a Failed Boot

This topic is organized around things to investigate or information to gather for different types of failures in some aspect of the booting process. Many of the techniques described here reference information provided in these topics:

- *Anatomy of an XC System Boot with xtbootsys* on page 27
- *SMW Daemons, Processes, and Logs* on page 13
- Commands Helpful in Troubleshooting a Boot, which is a list of topics that begins with *Check RSMS Daemons* on page 53.

A general technique for troubleshooting a boot is to start with the command that failed and look at appropriate logs to determine what might have caused that command to fail. The command that failed might be `xtbootsys`, which calls several other commands, and the real problem is that one of the called commands failed. One of the called commands may have failed because one of the normal daemons on the SMW is not running properly.

Another general technique is to check the ansible logs on a node (`/var/opt/cray/log/ansible/`) to find out what the relevant playbook tried to do compared to what it was supposed to do.

A standard diagnostic step is to ensure that all of the normal daemons on the SMW are running properly. Many of the commands called by `xtbootsys` depend on these daemons to be running on the SMW. Use one of these commands:

```
smw# /etc/init.d/rsms status

smw# systemctl status -l -n 99 rsms
```

If one of the rsms daemons is not running, start it using these commands:

```
smw# systemctl stop rsms
smw# systemctl start rsms
```

Each of the following topics describes a problem that can be encountered when booting an XC system and suggests ways to troubleshoot the problem.

xtcli status Fails

Procedure

1. Run the `xtcli status` command.

   ```
   smw# xtcli status p0
   ```

2. If `xtcli status` fails, check whether all of the rsms processes are running.

   ```
   smw# /etc/init.d/rsms status
   ```

   If any are not running, then resolve that problem before proceeding to the next step.

3. Run the `xtcli status` command again.

```
smw# xtcli status p0
```

xtbootsys Fails with xtbounce Error

About this task

If `xtbootsys` fails because of an error from `xtbounce`, the problem that caused `xtbounce` to fail must be resolved before the rest of the boot session will be able to succeed.

Procedure

1. Check the blade controller (BC) logs on the SMW.

   When `xtbounce` fails, it may be useful to look at the logs on the BC. These are sent to the SMW and will appear in the `/var/opt/cray/log/controller` directory structure. For example, for node c0-0c0s7n1, look in `/var/opt/cray/log/controller/c0-0/c0-0c0s7`.

   Whether looking on the SMW or directly on the BC, there is a BIOS file for each node on the blade and a messages file for general syslog info from the BC. On the SMW, the log files will have a date stamp on their file name in `/var/opt/cray/log/controller/c0-0/c0-0c0s7`.

   ```
   bios-n0-YYYYMMDD
   bios-n1-YYYYMMDD
   bios-n2-YYYYMMDD
   bios-n3-YYYYMMDD
   messages-YYYYMMDD
   ```

2. If BC logs not yet transferred, log in to BC and check them there.

   If the logs from the BC have not been transferred, then use `xtlogin` to log in to the BC and look in `/var/log`.

   ```
   crayadm@smw> xtlogin c0-0c0s7
   c0-0c0s7# cd /var/log
   ```

   On the BC, the log files will have more detail.

   ```
   bios_log.node0
   bios_log.node1
   bios_log.node2
   bios_log.node3
   messages
   ```

   There will probably be five older versions of these log files on the BC.

3. If one node on a blade has failed, compare its `bios_log` file with a node that did not fail.

4. Check the firmware on the blade with `xtzap` to ensure that it is current.

5. If firmware is stale, update it.

6. Power down and up the blade or ask a hardware person to reseat the memory, cards, or the entire blade in the chassis.

7. Run `xtbounce` again.

If the `xtbounce` problem cannot be resolved, it might be possible to disable some individual nodes or blades and then retry the `xtbounce`.

xtbootsys Fails with rtr Error

About this task

If `xtbootsys` fails because of an error from the `rtr` command, then the problem must be resolved before trying `xtbootsys` again. Routing can fail if too many components in key locations have been disabled.

Procedure

1. Run the `rtr` command to compute and place routes in the system.

```
smw# rtr -R p0
```

2. When the `rtr` command completes without any errors, then try xtbootsys again.

xtbootsys Fails with xtcablecheck Error

About this task

If `xtbootsys` fails because of an error from `xtcablecheck`, then fix the issue and confirm that `xtcablecheck` has no error.

Procedure

Run these commands to confirm that `xtcablecheck` has no error.

```
crayadm@smw> xtbounce p0

crayadm@smw> xtcablecheck p0
```

Boot or SDB Node Fails to PXE Boot

Procedure

1. Check the messages in the console log for the node.

2. Check the Ethernet connection between the boot or SDB node and the SMW.

Failure to PXE boot could be due to a missing Ethernet connection between the node and the SMW. This could be caused by a faulty cable, or the cable not plugged into the "admin" network (which should have connections from SMW, boot nodes, and SDB nodes), or an Ethernet switch with incorrect VLAN for one of the ports on the admin network.

3. Check that the SMW eth3 interface is configured as "up" and with the IP address 10.3.1.1.

   NOTE: SMW HA will show 10.3.1.2 on one SMW and 10.3.1.3 on the other, but the primary/first SMW will respond to 10.3.1.1. This example shows both 10.3.1.2 and 10.3.1.1 on the eth3 interface because this SMW is the first SMW of an SMW HA pair.

```
smw# wicked show eth3
eth3             up
      link:      #5, state up, mtu 1500
      type:      ethernet, hwaddr d4:ae:52:e6:9f:58
      config:    compat:suse:/etc/sysconfig/network/ifcfg-eth3
      leases:    ipv4 static granted
      addr:      ipv4 10.3.1.2/16 [static]
      addr:      ipv4 10.3.1.1/16
      route:     ipv4 10.128.0.0/14 via 10.3.1.254
```

   If that does show state "up" with the proper IP address, check `/etc/sysconfig/network/ifcfg-eth3`.

```
smw# cat /etc/sysconfig/network/ifcfg-eth3
BOOTPROTO='static'
IPADDR='10.3.1.1/16'
NAME='eth3 Boot node Network'
PREFIXLEN='16'
STARTMODE='auto'
USERCONTROL='no'
LINK_REQUIRED='no'
```

4. Check that the DHCP daemon (dhcpd) on the SMW is running properly, and if not, start it.

```
smw# systemctl status -l -n 99 dhcpd
```

   If dhcpd is not running, start it with this command:

```
smw# systemctl start dhcpd
```

5. Check that the TFTP daemon (atftpd) on the SMW is running properly, and if not, start it.

```
smw# systemctl status -l -n 99 atftpd
```

   If atftpd is not running, start it with this command:

```
smw# systemctl start atftpd
```

6. Check whether the firewall settings on SMW eth3 are preventing TFTP, and if they are, change them.

   If the firewall settings on SMW eth3 prevent TFTP, the PXE boot will fail. The SMW eth3 should be in FW_DEV_INT and not in FW_DEV_EXT in `/etc/sysconfig/SuSEfirewall2`. If eth3 was set to the external firewall zone using `yast2 firewall`, then use that command to change it back.

```
smw# yast2 firewall
```

   The admin network in the global config set should be set to `fw_external=false`.

```
smw# cfgset search -l advanced -t fw_external global
cray_global_net.settings.networks.data.admin.fw_external: false
```

If it is set to true, that means it is in the external zone for the firewall. Change it to false.

```
smw# cfgset update -m interactive -l advanced -s cray_global_net global
Global Networking Menu [default: save & exit - Q] $ 1
Global Networking Menu [default: configure - C] $ C
...
cray_global_net.setting.networks
[<cr>=set 5 entries, +=add an entry, ?=help, @=less] $ 1i*
...
cray_global_net.setting.networks.data.admin.fw_external
[<cr>=set 'true', <new value>, ?=help, @=less] $ false
```

Remember to press **Enter** several times to set the new value and save changes to the config set.

7. Check whether the boot or SDB node has the correct boot image assigned to it.

   Ensure that the assigned kernel parameters do not include a setting for Netroot. Only login and compute nodes can use the Netroot kernel parameter when they have Netroot boot images assigned.

```
smw# cnode list c0-0c0s0n1
```

8. Check whether the boot image is too large for PXE boot.

   There is a limit on the size of the boot image that will successfully PXE boot. If this is a problem, `xtbootsys` will display an error message "Initramfs too big for PXE boot." That limit can be adjusted by changing a setting in `/opt/cray/hss/default/etc/bm.ini`.

9. Check the status of the cray-ids-service on the SMW.

   The IMPS Distribution Service (IDS) distributes the config set data to nodes on the XC system. The distributed I/O daemon (diod) does I/O forwarding for IDS. The output from `systemctl status` may indicate that cray-ids-service had a problem starting the diod daemon or the diod daemon may no longer be running. IDS requires that diod be running for config sets to be distributed to the CLE nodes.

```
smw# systemctl status -l -n 99 cray-ids-service
cray-ids-service.service - cray-ids-service server
   Loaded: loaded (/usr/lib/systemd/system/cray-ids-service.service; disabled)
   Active: active (running) since Tue 2016-06-21 22:34:52 CDT; 3 weeks 0 days
ago
 Main PID: 53484 (diod)
   CGroup: /system.slice/cray-ids-service.service
           +-53484 /usr/sbin/diod -U root --export-opts ro --allsquash --no-a...
```

10. Check which upstream IDS servers are listed in the IDS kernel parameter for the node.

    For the boot and SDB nodes (as tier1 nodes), the only node listed in the IDS kernel parameter should be `ids=10.3.1.1`, which is the SMW's IP address on eth3. This is the server from which the boot and SDB nodes will mount the config set.

```
boot# cat /proc/cmdline
BOOT_IMAGE=net0:/opt/tftpboot/elilo.config/c0-0c0s0n1/
bzImage-3.12.51-52.31.1_1.0600.9146-cray_ari_s
earlyprintk=serial,115200 load_ramdisk=1 ramdisk_size=80000
console=ttyS0,115200n8
bootproto=ipog oops=panic elevator=noop pcie_ports=native iommu=on
intel_iommu=off bad_page=panic
```

```
apei_disable hest_disable erst_disable ghes_disable cgroup_disable=memory
audit=0
sessionid=p0-20160621t214422 sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 tier1=1 NIMS_GROUP=service ids=10.3.1.1 config_set=p0
crayname=c0-0c0s0n1
```

Possible Problem with Boot Image Assignment

Have all nodes been assigned a boot image by NIMS?

Show the boot configuration used by the boot manager. This will list all of the boot images assigned via NIMS.

> **NOTE:** This command requires that the state_manager and nimsd be running.

```
crayadm@smw> xtcli part_cfg show
Network topology: class 0
=== part_cfg ===
------------------
[partition]: p0: enable (noflags|)
[members]: c0-0
[boot]: c0-0c0s0n1:halt,c0-0c1s0n1:halt
[sdb]: c0-0c0s1n1:halt,c0-0c1s1n1:halt
[NIMS_image 0]: /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 1]: /var/opt/cray/imps/boot_images/service_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 2]: /var/opt/cray/imps/boot_images/dal_cle_6.0.UP02-
build6.0.2042_centos_6.5-created20161215.cpio
[NIMS_image 3]: /var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
[NIMS_image 4]: /var/opt/cray/imps/boot_images/fio-service_cle_6.0.UP02-
build6.0.2042_sles_12-created20161215.cpio
```

Is there a boot image in this list that is unexpected? If the NIMS_image 0 has a null value, that means that some node does not have any boot image assigned to it. This might show up as a "Boot manager CPIO package error" from the `xtcli boot` command.

xtbootsys Exits After Failure to Boot the Boot and SDB Nodes

About this task

If xtbootsys exits after failing to boot the boot and SDB nodes, it usually means that the `xtcli boot` command used to boot the nodes had a failure. This might be due to a missing boot image.

Procedure

Check for a missing boot image.

See the following topics for information about how to do this:

- *Check Which Boot Images Have Been Assigned* on page 59
- *Check Node NIMS Group, Boot Image, and Kernel Parameter Assignment* on page 59
- *Check Which Boot Images Exist on the System* on page 61

xtbootsys Exits After Timeout While Booting the Boot and SDB Nodes

About this task

If `xtbootsys` exits after a timeout while booting the boot and SDB nodes, this means that the `xtcli boot` command was able to send the boot image to the node. However, it probably means that the node has an issue, which will be displayed on the console log for this node.

A timeout is likely to be hit if the node has dropped into the DEBUG shell. The console log will clearly indicate that the node is in the DEBUG shell and what reason caused it to enter the DEBUG shell. For example:

```
2016-06-16T15:59:36.173243-05:00 c0-0c0s9n1 DEBUG SHELL: reason for being in DEBUG
shell; exit will resume
2016-06-16T15:59:36.173255-05:00 c0-0c0s9n1 DEBUG SHELL
2016-06-16T15:59:36.173486-05:00 c0-0c0s9n1 :/ #
```

This procedure investigates why this command entered the DEBUG shell.

Procedure

1. Check whether the node has the DEBUG=true kernel parameter set.

   If the node has the DEBUG=true kernel parameter set, then that is why it entered the DEBUG shell and the console-log reason will show one of the debug breakpoints in the `/init` script of the image root. While in the DEBUG shell, explore the state of things on the node, then enter **`exit`** so that `/init` can continue to do steps until the next breakpoint, which will enter the DEBUG shell again and show the next reason for being in the DEBUG shell.

   This node has DEBUG set:

   ```
   smw# cnode list c0-0c0s0n1 | grep DEBUG
   c0-0c0s0n1  service  service  /var/opt/cray/imps/boot_images/
   service_cle_6.0.UP01-build6.0.96_sles_12-created20160705.cpio
   -           sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254 tier1=1
   hsn_ipv4_mask=255.252.0.0
   DEBUG=true hsn_ipv4_net=10.128.0.0 NIMS_GROUP=service ids=10.3.1.1 config_set=p0
   ```

   This node does not have DEBUG set:

   ```
   smw# cnode list c0-0c0s0n1 | grep DEBUG
   smw#
   ```

2. Check whether the node failed when `/init` called `cray-ansible`.

   If the node had a failure when `/init` called `cray-ansible`, then it displays messages to the console related to the last Ansible task that failed and enters the DEBUG shell. This needs administrator interaction on the console.

   a. Connect to the node to interact with the DEBUG shell.

      ```
      smw# xtcon c0-0c0s0n1
      DEBUG#
      ```

   b. Run commands to explore the state of the node.

Be aware that systemd has not started at this point, so several daemons will not be running, and any `systemctl status` commands will probably fail.

c. If a config set change is need to resolve the Ansible error, make the change on the SMW and then test it on the node by running cray-ansible again.

```
DEBUG# /etc/init.d/cray-ansible start
```

If cray-ansible succeeds, it will show a success message. If cray-ansible fails, inspect the Ansible error log and repeat the process to change the config set.

3. Exit the DEBUG shell.

```
DEBUG# exit
```

Once the DEBUG shell is exited, cray-ansible will be run again. If it succeeds, then `/init` will continue and move to systemd startup, then cray-ansible will run again in the booted phase. If it fails, then the console will return to the DEBUG shell again for another attempt to resolve the problem.

NOTE: To disconnect an `xtcon` session from the console of a node, type `^]` to quit.

xtbootsys Waits for Input After Timeout While Booting the Boot and SDB Nodes

About this task

If `xtbootsys` detects that the service nodes have exceeded the boot timeout, then `xtbootsys` will display a prompt asking whether to pause the boot. A response is expected within 300 seconds or the boot will continue with the next task in the boot automation file.

Procedure

1. In the `xtbootsys` screen, enter **?** to see which nodes failed.

```
##########################################################
Wed Jun 15 16:05:04 CDT 2016
1 service node failed to boot, you can ...
    enter a 'q' to quit xtbootsys
    enter a 'c' to continue xtbootsys
    enter a 'p' to pause xtbootsys
    enter a '?' to see which nodes failed
    do nothing and this menu will time out in 300 seconds and xtbootsys will
continue.
?
```

If no response is provided in within 300 seconds, then this message will be displayed and the boot will continue with the next task in the boot automation file.

```
Nothing was chosen in the allotted time, xtbootsys will continue...
```

Depending on what services are provided by the failed service nodes, the next steps may fail. If service nodes failed to boot, then the boot of compute nodes afterwards may fail and appear to be the problem, but it is this earlier service node failure that needs to be fixed first.

2. In the `xtbootsys` screen, enter **p** to pause `xtbootsys` and investigate why those nodes had a problem.

3. If the problem on the failed nodes can be resolved, then return to `xtbootsys` and enter **c** to continue to the next step in the boot automation file.

xtbootsys Never Begins to Boot Service Nodes

About this task

If xtbootsys never begins to boot service nodes, except for the boot and SDB nodes, it is because there is a problem with the `xtcli boot all_serv` command used to boot the service nodes.

Procedure

1. Check whether proper boot images are assigned to the service nodes.

   It is possible that different service nodes have different boot images, so check all of them. There are several ways to do this.

Option	Description
all nodes	Check all boot image assignments (this is probably too much output since it will show all nodes in the system). `smw# cnode list`
just the nodes that failed	Check the list of nodes that failed. `smw# cnode list c0-0c0s7n1 c0-0c1s3n2`
all service nodes	Check all nodes in the "service" NIMS group. `smw# cnode list --filter group=service`
all login nodes	Check all nodes in the "login" NIMS group. Was it only login nodes that failed to boot? `smw# cnode list --filter group=login`
all DAL nodes	Check all nodes in the "dal" NIMS group. Was it only DAL nodes that failed to boot? `smw# cnode list --filter group=dal`
DataWarp nodes with FIO SSDs	Check all nodes in the "fio-service" NIMS group. Was it only DataWarp nodes with Fusion I/O SSDs using the fio-service boot image that failed to boot? `smw# cnode list --filter group=fio-service`
other	There could be other custom NIMS groups in use on the machine. Check those as well.

2. Check whether the boot node did an NFS mount of `/var/opt/cray/imps/boot_images` from the SMW.

   When the boot node is up, any requests to boot nodes using the HSN will send a request from the SMW to the boot node daemon (bnd) on the boot node. The boot node NFS mounts

the `/var/opt/cray/imps/boot_images` directory from the SMW to its own `/var/opt/cray/imps/boot_images` mount point so that bnd can access all of the required boot images. bnd needs this to be able to use the boot images for boots over the HSN.

a. Is `/etc/exports` incorrect on the SMW?

```
smw# grep boot_images /etc/exports
/var/opt/cray/imps/boot_images
10.0.0.0/8(secure,rw,no_subtree_check,no_root_squash)
```

b. Is the nfsserver daemon running on the SMW? If not, start it.

```
smw# systemctl status -l -n 99 nfsserver
nfsserver.service - LSB: Start the kernel based NFS daemon
   Loaded: loaded (/etc/init.d/nfsserver)
   Active: active (running) since Tue 2016-06-21 12:54:58 CDT; 3 weeks 1
days ago
   CGroup: /system.slice/nfsserver.service
           +-4025 /usr/sbin/rpc.idmapd -p /var/lib/nfs/rpc_pipefs
           +-4066 /usr/sbin/rpc.mountd
```

If nfsserver is not running, start it with this command.

```
smw# systemctl start nfsserver
```

c. Is the nfsd daemon running on the SMW?

```
smw# ps -ef | grep nfs
ps -ef | grep nfs
root    4025    1  0 Jun21 ?    00:00:00 /usr/sbin/rpc.idmapd -p /var/lib/
nfs/rpc_pipefs
root    4135    2  0 Jun21 ?    00:00:00 [nfsd4]
root    4136    2  0 Jun21 ?    00:00:00 [nfsd4_callbacks]
root    4162    2  0 Jun21 ?    00:00:00 [nfsd]
root    4163    2  0 Jun21 ?    00:00:02 [nfsd]
root    4164    2  0 Jun21 ?    00:00:00 [nfsd]
root    4165    2  0 Jun21 ?    00:00:01 [nfsd]
```

d. Is `/etc/fstab` incorrect on the boot node?

```
boot# grep boot_images /etc/fstab
smw:/var/opt/cray/imps/boot_images /var/opt/cray/imps/boot_images nfs ro 0 0
```

3. Check whether the boot node daemon (bnd) is running on the boot node.

```
boot# ps -ef | grep bnd
```

4. Check bnd logs on the SMW.

Output from bnd is sent to the SMW in `/var/opt/cray/log/p0-current/messages-`*`YYYYMMDD`* and can be found by searching for "bnd" in that file.

```
smw# grep bnd /var/opt/cray/log/p0-current/messages-20160621
```

Look for the section that looks similar to this for the boot image being used:

```
2016-06-21T21:58:17.708852-05:00 c0-0c0s0n1 bnd 11958 p0-20160621t214422
[sys_boot@34] ***** HSN Booting 5 nodes using DEFAULT in
```

```
/var/opt/cray/imps/boot_images/service_cle_6.0.UP01-build6.0.96_sles_12-
created20160614.cpio
```

There will be more messages after this point that describe how many nodes were being booted and what the status code is from each.

xtbootsys Never Begins to Boot Compute Nodes

About this task

If `xtbootsys` never begins booting compute nodes, there is a problem with the `xtcli boot all_comp` command used to boot the compute nodes.

Procedure

1. Check whether proper boot images are assigned to the compute nodes.

   It is possible that different compute nodes have different boot images, so check all of them. There are several ways to do this.

Option	Description
all nodes	Check all boot image assignments (this is probably too much output since it will show all nodes in the system). `smw# cnode list`
just the nodes that failed	Check the list of nodes that failed. `smw# cnode list c0-0c0s9n3 c0-0c2s14n2`
all compute nodes	Check all nodes in the "compute" NIMS group. `smw# cnode list --filter group=compute`
other	There could be other custom NIMS groups in use on the machine. Check those as well.

2. Check bnd logs on the SMW.

   Output from bnd is sent to the SMW in `/var/opt/cray/log/p0-current/messages-YYYYMMDD` and can be found by searching for "bnd" in that file.

   ```
   smw# grep bnd /var/opt/cray/log/p0-current/messages-20160621
   ```

   Look for the section that looks similar to this for the boot image being used:

   ```
   2016-06-21T22:16:51.290601-05:00 c0-0c0s0n1 bnd 13100 p0-20160621t214422
   [sys_boot@34] ***** HSN Booting 1 nodes using DEFAULT in
   /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
   build6.0.96_sles_12-created20160615.cpio
   ```

   There will be more messages after this point that describe how many nodes were being booted and what the status code is from each.

cray-ansible Fails in Init Phase on any Node

About this task

If cray-ansible fails during the init phase, it will display this message on the node's console.

```
cray-ansible: /etc/ansible/site.yaml completed in init - FAILED
```

If the node had a failure when `/init` called cray-ansible, then it displays messages to the console related to the last Ansible task that failed and enters the DEBUG shell. This needs administrator interaction on the console.

Procedure

1. Connect to the node to interact with the DEBUG shell.

   ```
   smw# xtcon c0-0c0s0n1
   DEBUG#
   ```

2. Run commands to explore the state of the node.

   Be aware that systemd has not started at this point, so several daemons will not be running, and any `systemctl status` commands will probably fail.

3. Identify which Ansible task failed.

   Look at the Ansible logs on the node in `/var/opt/cray/log/ansible`. See *cray-ansible and Ansible Logs on a CLE Node* on page 51 for names and contents of the log files.

   In this example Ansible log, the task that failed is "task motd, release."

   ```
   2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
   ****************************************
   2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true,
   "cmd": "grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print
   $2}'", "delta": "0:00:00.002536",
   "end": "2016-01-17 12:15:27.471384", "rc": 0, "start": "2016-01-17
   12:15:27.468848", "stderr": "",
   "stdout": "6.0.UP02", "warnings": []}
   ```

   In this example Ansible log, the task that failed is "task motd, release."

   ```
   2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
   ****************************************
   2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true,
   "cmd": "grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print
   $2}'", "delta": "0:00:00.002536",
   "end": "2016-01-17 12:15:27.471384", "rc": 0, "start": "2016-01-17
   12:15:27.468848", "stderr": "",
   "stdout": "6.0.UP02", "warnings": []}
   ```

4. Determine which Ansible play contained the task that failed.

   The location of a task can be found in the Ansible plays. Because the example shows that the failed task has the name "task motd, release," search for that name in the locations that have Ansible plays:

   - `/etc/ansible` has Ansible plays from the software image root

- `/etc/opt/cray/config/current` has Ansible plays from the current CLE config set, which may include site-local plays
- `/etc/opt/cray/config/global` has Ansible plays from the global config set, which may include site-local plays

```
boot# grep -Rn "task motd, release" /etc/ansible \
/etc/opt/cray/config/current/ansible /etc/opt/cray/config/global/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

The grep output shows that the file containing this task is `/etc/ansible/roles/cle_motd/tasks/motd.yaml`. Look at line 15 in that file to determine what that chunk of code is doing.

5. Check the cray_system facts for this node.

Look at the Ansible code in the identified task/play. Check that the boot_node, backup_boot_node, sdb_node, and backup_sdb_node are correct, and that the nims_group is the expected one for this node (which is related to which image might be used on the node). Use this command to display the cray_system facts for this node.

```
DEBUG# /etc/ansible/facts.d/cray_system.fact
{"topology_class": 0, "host_type": "admin", "node_groups": [], "sdb_node": 38,
"platform": "service", "max_torus_dimension": [0, 0, 9], "max_node_id": 39,
"nid": 1,
"in_init": false, "sessionid": "p0-20160621t214422", "hostid": "c0-0c0s0n1",
"standby_node": false, "num_sys_nodes": 16, "max_sys_nodes": 16, "roles":
["boot"],
"is_cray_blade": true, "uses_systemd": true, "cname": "c0-0c0s0n1",
"sys_nodes": [0, 1, 2, 3, 4, 5, 6, 7, 32, 33, 34, 35, 36, 37, 38, 39],
"boot_node": 1,
"nims_group": ["service"], "backup_sdb_node": 5}
```

6. Check the kernel parameters that were passed to the node.

These should match what was assigned to the node with NIMS (using the `cnode list` command), but will also show the nodes (represented by their IP address) assigned for the `ids` kernel parameter.

```
DEBUG# cat /proc/cmdline
earlyprintk=serial,115200 load_ramdisk=1 ramdisk_size=80000 console=ttyS0,
115200n8 bootproto=ipog oops=panic elevator=noop pcie_ports=native iommu=on
intel_iommu=off bad_page=panic apei_disable hest_disable erst_disable
ghes_disable
cgroup_disable=memory audit=0  sessionid=p0-20160712t104308
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
NIMS_GROUP=login netroot=login-large_cle_6.0.UP02-build6.0.2042 _sles_12-
created20161215
ids=10.128.1.134,10.128.0.79,10.128.0.78 config_set=p0
```

7. Check whether the nodes in the ids kernel parameter are not booted.

If none of the nodes (represented by their IP address) in the ids kernel parameter are booted, then this node will fail to boot. Shift the analysis to those nodes instead of this node. The tier1 nodes (boot and SDB nodes) will have the SMW eth3 10.3.1.1 IP address for IDS. The tier2 nodes will have the IP addresses on the HSN of the tier1 nodes. All other nodes are tier3 nodes, which will have IP addresses on the HSN of the tier2 nodes.

8. If a config set change is needed to resolve the Ansible error, make that change on the SMW and test it on the node.

   a. Update the config set on the SMW.

   ```
   smw# cfgset update -m interactive CONFIGSET
   ```

   Substitute the applicable config set (global, p0, etc.) for *CONFIGSET*, depending on which services/parameters need to be changed.

   b. Run cray-ansible on the node to test the config set change there.

   ```
   node# /etc/init.d/cray-ansible start
   ```

   If cray-ansible succeeds, it will show a success message. If cray-ansible fails, return to step 3.

9. Exit the DEBUG shell:

   ```
   DEBUG# exit
   ```

   Once the DEBUG shell is exited, cray-ansible will be run again. If cray-ansible succeeds, then `/init` will continue and move to systemd startup, then cray-ansible will run again in the booted phase. If cray-ansible fails, then the console will return to the DEBUG shell again for another attempt to resolve the problem.

   NOTE: To disconnect an `xtcon` session from the console of a node, type `^]` to quit.

cray-ansible Fails in Booted Phase on Any Node

About this task

If a node fails because of a cray-ansible failure in the booted phase, an error message like this appears:

```
cray-ansible: /etc/ansible/site.yaml completed in booted - FAILED
```

When this happens, it is difficult to know how many of the Ansible plays were executed prior to the Ansible play that failed. Try this procedure:

Procedure

1. Get access to the failed node.

   If the node has partially booted, then it might have booted far enough to start the sshd daemon before hitting a failure, thereby making it possible to log in using `ssh`. Otherwise, use `xtcon` to connect.

Option	Description

If `ssh` available	Use `ssh` to log in from the SMW to the boot or SDB node.
	`smw# ssh boot`
	`smw# ssh sdb`
	Then from that node, log in to the failed node.
	`boot# ssh c0-1c2s7n2`
	`sdb# ssh c1-2c0s15n3`

Option	Description
If `ssh` not available	Use `xtcon` to connect from the SMW to the failed node. `smw# xtcon c0-1c2s7n2` Note: To disconnect an `xtcon` session from the console of a node, type **`^]`** to quit.

Now commands can be run on the failed node to explore the state of the node.

2. Identify which Ansible task failed.

   Look at the Ansible logs on the node in `/var/opt/cray/log/ansible`. See *cray-ansible and Ansible Logs on a CLE Node* on page 51 for names and contents of the log files.

3. Check the cray_system facts for this node.

   Look at the Ansible code in the identified task/play. Check that the boot_node, backup_boot_node, sdb_node, and backup_sdb_node are correct, and that the nims_group is the expected one for this node (which is related to which image might be used on the node). Use this command to display the cray_system facts for this node.

```
node# /etc/ansible/facts.d/cray_system.fact
{"topology_class": 0, "host_type": "admin", "node_groups": [], "sdb_node": 38,
"platform": "service", "max_torus_dimension": [0, 0, 9], "max_node_id": 39,
"nid": 1,
"in_init": false, "sessionid": "p0-20160621t214422", "hostid": "c0-0c0s0n1",
"standby_node": false, "num_sys_nodes": 16, "max_sys_nodes": 16, "roles":
["boot"],
"is_cray_blade": true, "uses_systemd": true, "cname": "c0-0c0s0n1",
"sys_nodes": [0, 1, 2, 3, 4, 5, 6, 7, 32, 33, 34, 35, 36, 37, 38, 39],
"boot_node": 1,
"nims_group": ["service"], "backup_sdb_node": 5}
```

4. Check the kernel parameters that were passed to the node.

   These should match what was assigned to the node with NIMS (using the `cnode list` command), but will also show the nodes assigned for the `ids` kernel parameter.

```
node# cat /proc/cmdline
earlyprintk=serial,115200 load_ramdisk=1 ramdisk_size=80000
console=ttyS0,115200n8
bootproto=ipog oops=panic elevator=noop pcie_ports=native iommu=on
intel_iommu=off
bad_page=panic apei_disable hest_disable erst_disable ghes_disable
cgroup_disable=memory
audit=0  sessionid=p0-20160712t104308 hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0
sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254 NIMS_GROUP=login
netroot=gin-large_cle_6.0.UP01-build6.0.96_sles_12-created2016061
ids=10.128.1.134,10.128.0.79,10.128.0.78 config_set=p0
```

5. If a config set change is needed to resolve the Ansible error, make that change on the SMW and test it on the node.

   a. Update the config set on the SMW.

```
smw# cfgset update -m interactive CONFIGSET
```

Substitute the applicable config set (global, p0, etc.) for *CONFIGSET*, depending on which services/parameters need to be changed.

b. Run cray-ansible on the node to test the config set change there.

```
node# /etc/init.d/cray-ansible start
```

If cray-ansible succeeds, it will show a success message. If cray-ansible fails, return to step 2.

Node Fails to Mount Local Storage

Procedure

1. Log in to the node using `ssh` or `xtcon`.

   For details about how to do this, see step 1 of *cray-ansible Fails in Booted Phase on Any Node* on page 82.

2. Check what disk devices are available on the node.

```
node# lsscsi
node# fdisk -l
```

3. If some disk devices are missing, ensure that the node has an FC or SAS card by inspecting `lspci` output.

```
node# lspci
```

This output can be very verbose. If the vendor of the FC or SAS card is known, search for that in the output.

```
node# lspci | grep -i qlogic
05:00.0 Fibre Channel: QLogic Corp. ISP8324-based 16Gb Fibre Channel to PCI
Express Adapter (rev 02)
05:00.1 Fibre Channel: QLogic Corp. ISP8324-based 16Gb Fibre Channel to PCI
Express Adapter (rev 02)
```

4. If the FC or SAS card is present but no devices are seen, check the cabling and the zoning.

   a. Check the cabling between the card and the FC or SAS switch.

   b. Check the zoning on the FC or SAS switch to ensure that the node is permitted to see the storage.

Node Fails to Mount NFS File System

Procedure

1. Check that the proper entry is in `/etc/fstab` on the node.

2. If the node is supposed to automount the NFS file system, investigate the following.

   a. Are the proper files in `/etc/auto.master` and `/etc/auto.master.d` on the node?

   For information about how to configure automount files on a DVS node using Simple Sync, see "Reconfigure DVS for an External NFS Server" in *XC™ Series DVS Administration Guide* (S-0005).

b. Are these files being distributed using the cray_simple_sync service?

Look in the `/var/opt/cray/imps/config/sets/p0/files/simple_sync` directory on the SMW.

- Is this node listed there (by cname) in the `hardwareid` subdirectory? If so, look in `hardwareid/`*`NODE_CNAME`*`/files` to see if the proper files are there.
- Is this node supposed to be in one of the node groups in the `nodegroups` subdirectory? If so, look in `nodegroups/`*`NODEGROUP`*`/files` to see if the proper files are there.

c. If the proper files are in simple_sync, is this node a member of that node group in the cray_node_groups service?

This command will search for a node group (from the simple_sync directory structure) with the name *NODEGROUP*.

```
smw# cfgset search -t NODEGROUP -s cray_node_groups p0
```

3. Is the NFS server exporting the NFS file system to this node?

Option	Description
external NFS server	If the NFS server is external to the XC system, then look in `/etc/exports` on the external NFS server.
internal NFS server	If he NFS server is internal to the XC system, then look for the NFS section of the cray_simple_shares service in the CLE config set, and check the guidance text for the displayed settings. `smw# cfgset search -l advanced -s cray_simple_shares p0 | grep NFS`

This node might be DVS-projecting the NFS file system to other nodes. If so, those other nodes will fail to DVS-mount this file system.

Node Fails to Mount Direct-attached Lustre (DAL)

Procedure

1. Check the gni number. Is it wrong?

The command to mount the DAL file system includes the LNet nid of the Lustre MGS, which consists of the nid number, followed by an '@' symbol, followed by the LNet name, such as gni or gni1. For example, if the nid number of the DAL MGS is nid00009, and the LNet name is gni1, then the LNet nid is 9@gni1.

a. Find the LNet name being used in the CLE config set.

The LNet name that has been added to the configuration data can be found by searching for "gni" in the CLE config set (p0 in this example) on the SMW.

```
smw# cfgset search -t gni -l advanced -s cray_lnet p0
# 2 matches for 'gni' from cray_lnet_config.yaml
#-----------------------------------------------------------------------
cray_lnet.settings.local_lnet.data.lnet_name: gni4
cray_lnet.settings.flat_routes.data.o2ib.src_lnet: gni4
```

b. Find the LNet nid of the Lustre client node.

```
node# lctl list_nids
```

c. Find the LNet nid of the DAL MGS node.

   The gni number should be the same as the client.

```
mgs# lctl list_nids
```

d. From the client node, ping the DAL MGS node by its LNet nid, and then from the MGS node, ping the client node by its LNet nid.

   In this example, the LNet nid of the MGS iss 9@gni1, and the LNet nid of the client node is 30@gni1.

```
node# lctl ping 9@gni1
mgs# lctl ping 30@gni1
```

   If these two nodes cannot ping each other, this might be due to different gni numbers or to some other CLE-side problem.

2. Check to see if the DAL file system is started.

If the user knows the name of the DAL file system, e.g. "dal", then instead of the "-a" option to lustre_control they can use "lustre_control status -f $FS_NAME", where FS_NAME is the name of the DAL file system.

```
boot# module load lustre-utils
boot# lustre_control status -a
```

If the name of the DAL file system is known (e.g., "dal"), then use the `-f` option with the DAL file system name instead of the `-a` option. This command will ssh from the boot node to the Lustre servers (MGS, MDS, OSS), check whether the targets (MGT, MDT, OST) are mounted on the nodes, and report this information in a table.

```
boot# lustre_control status -f $dal
```

The lustre_control commands to start the DAL file system and mount it on clients should be in the boot automation file. See *XC™ Series Software Installation and Configuration Guide* (S-2559) for these commands.

3. Check Ansible logs and config set data.

a. Check the config set data on the SMW to ensure that it is configured correctly.

b. Look at Ansible logs on the client node.

   Search for "lustre_client" in `/var/opt/cray/log/ansible/sitelog-booted`. Look at the tasks, to see if the tasks that supposed to mount the Lustre file systems are being skipped.

c. Look at the config set data on the client node in `/etc/opt/cray/config/current/config/cray_lustre_client_config.yaml` and compare it to the config data on the SMW.

   If config data is different on the SMW than on the client node, there may be a problem with the config set cache not being updated. See *Check cray-cfgset-cache Daemon* on page 54.

Node Fails to Mount External Lustre File System

Procedure

1. Check to see if the external Lustre server is up and serving the file system.

2. Check to see if the LNet routes are set up properly on the external Lustre servers.

   The external Lustre servers live on an InfiniBand network, so they have LNet nids with an LNet name starting with 'o2ib' (e.g., 10.149.0.1@o2ib). They need a route to reach the CLE client nodes that are only on the HSN network and thus have only LNet nids with an LNet name starting with 'gni' (e.g. 30@gni1). Thus, the Lustre servers need a route from LNet nids on o2ib to LNet nids on gni. This route will be through the CLE LNet router nodes, which have both InfiniBand and HSN interfaces, and thus have both gni and o2ib LNet nids. If these routes are not set up correctly on the Lustre servers, it might look like there is an error with routes on the CLE nodes because they cannot ping the servers, but the problem might be on the server side. For more information about setting up LNet routes, see XC™ Series Lustre® Administration Guide.

3. Check the gni number. Is it wrong?

   a. Find the LNet name being used in the CLE config set.

      The LNet name that has been added to the configuration data can be found by searching for "gni" in the CLE config set (p0 in this example) on the SMW.

      ```
      smw# cfgset search -t gni -l advanced -s cray_lnet p0
      # 2 matches for 'gni' from cray_lnet_config.yaml
      #-----------------------------------------------------------------------
      cray_lnet.settings.local_lnet.data.lnet_name: gni4
      cray_lnet.settings.flat_routes.data.o2ib.src_lnet: gni4
      ```

   b. Find the LNet nid of the Lustre client node.

      The LNet nid of a node consists of the node nid number, followed by an '@' symbol, followed by the LNet name, such as gni or gni1.

      ```
      node# lctl list_nids
      ```

   c. Find the LNet nid of the LNet router node.

      LNet routers have two or more LNet nids that will be listed in the output of this command. One of them should have the same LNet name as the client.

      ```
      lnet# lctl list_nids
      10.149.0.34@o2ib
      33@gni99
      33@gni1
      ```

   d. From the client node, ping the router node by its LNet nid, and then from the router node, ping the client node by its LNet nid.

      In this example, the LNet nid of the router is 33@gni1, and the LNet nid of the client node is 30@gni1.

      ```
      node# lctl ping 33@gni1
      lnet# lctl ping 30@gni1
      ```

      If these two nodes cannot ping each other, this might be due to different gni numbers or to some other CLE-side problem.

e. If these two pings work, then on the LNet router, use `lctl ping` to ping the external Lustre server by its LNet nid.

   The LNet nid(s) of the MGS(s) can be found in the `cray_lustre_client.settings.client_mounts.data.<key>.mgs_lnet_nids` configuration setting. In this example, the external Lustre server has LNet nid 10.149.0.1@o2ib.

```
lnet# lctl ping 10.149.0.1@o2ib
```

Node Fails to Mount DVS-projected File System

Procedure

1. Check the file system mount points of each DVS server listed for that file system.

   a. On one of the nodes that is mounting a file system from a DVS server, check `/etc/fstab` to see which DVS servers are listed for the file system.

   b. On each DVS server, check `/etc/fstab` to see if it has the file system mounted properly.

   c. On each DVS server, check whether the file system that is supposed to be DVS-projected is actually mounted on the DVS server.

```
dvs# mount
```

2. Are all of the DVS servers in the node's `/etc/fstab` unavailable?

   At least one should be available to mount the file system.

Corrupt File System on Boot or SDB Node

About this task

If one of the boot or SDB nodes that mount file systems from the boot RAID has file system corruption, there are a few different ways to repair the file system.

See *XC™ Series System Administration Guide* (S-2393) for procedures to repair Btrfs and XFS file systems.

Procedure

1. Unmount the file system from the node and repair it on that node, if possible.

2. Shut down the node and repair it from the SMW, if repairing it on the node is not possible.

Check the Duties of a Broken Service Node

To find out which duties a broken service node performs for a system, search the CLE config set. This example finds all of the places node c0-0c2s3n2 is mentioned in config set p0.

```
smw# cfgset search -t c0-0c2s3n2 -l advanced p0
```

If the node has network interfaces, results from cray_net will be displayed.

If the node is part of a node group, results from cray_node_groups will be displayed.

If the node is listed explicitly as a client or server for a feature, results from other config set services will be displayed.

Check for HSS and Config Set Agreement on Duties of Boot and SDB Nodes

If there is a mismatch between the boot and SDB nodes in the output from these two commands, there will be issues with the boot.

This first command shows that the Hardware Supervisory System (HSS) database recognizes two nodes as boot and two as SDB (because boot node failover and SDB node failover are both enabled).

```
smw# xtcli part_cfg show
Network topology: class 0
=== part_cfg ===
------------------
[partition]: p0: enable (noflags|)
[members]: c0-0
[boot]: c0-0c0s0n1:halt,c0-0c1s0n1:halt
[sdb]: c0-0c0s1n1:halt,c0-0c1s1n1:halt
[NIMS_image 0]: /var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio
[NIMS_image 1]: /var/opt/cray/imps/boot_images/service_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio
[NIMS_image 2]: /var/opt/cray/imps/boot_images/dal_cle_6.0.UP01-
build6.0.96_centos_6.5-created20160705.cpio
[NIMS_image 3]: /var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio
[NIMS_image 4]: /var/opt/cray/imps/boot_images/fio-service_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio
================
```

This search command (applied once for the boot node and again for the SDB node) searches the CLE config set for the host IDs of those two nodes.

```
smw# cfgset search -t bootnode -s cray_net p0 | grep hostid
cray_net.settings.hosts.data.bootnode.hostid: c0-0c0s0n1

smw# cfgset search -t sdbnode -s cray_net p0 | grep hostid
cray_net.settings.hosts.data.sdbnode.hostid: c0-0c0s1n1
cray_net.settings.hosts.data.backup_sdbnode.hostid: c0-0c0s9n2
```

Notice that the first "cfgset search" command shows only a single boot node, unlike the "xtcli part_cfg show" command which shows two boot nodes. The second "cfgset search" command shows a backup_sdbnode host ID that does not match the second SDB node cname from the "xtcli part_cfg show" command.

This is an example of mismatched data between the HSS database and the config set. The data needs to be corrected to match.

Node with Network Interface not Accessible over that Network

About this task

This procedure investigates whether the network interface is configured with correct networking information, is marked as "up," has a cable connected, sees network traffic, and sees network traffic on the correct network based on IP address and netmask.

Procedure

1. Find the node's network interface.

   Search for that node (c1-0c0s7n1 in the example) in cray_net, and then check the displayed information.

   ```
   smw# cfgset search -t c1-0c0s7n1 -s cray_net -l advanced p0
   ```

   If the node has a network interface, it will be listed as being on a network.

2. Ensure that the IP address for the network interface is on the network with the given netmask.

   Search for that network (network42 in the example) in cray_net, and then check the displayed information.

   ```
   smw# cfgset search -t network42 -s cray_net -l advanced p0
   ```

3. Is the network interface (eth3 in the example) "up" and configured with IP address 10.3.1.1?

   ```
   smw# wicked show eth3
   eth3             up
         link:      #5, state up, mtu 1500
         type:      ethernet, hwaddr d4:ae:52:e6:9f:58
         config:    compat:suse:/etc/sysconfig/network/ifcfg-eth3
         leases:    ipv4 static granted
         addr:      ipv4 10.3.1.2/16 [static]
         addr:      ipv4 10.3.1.1/16
         route:     ipv4 10.128.0.0/14 via 10.3.1.254
   ```

4. If the network interface does not show state "up" with the proper IP address, check `/etc/sysconfig/network/ifcfg-eth3`.

   ```
   smw# cat /etc/sysconfig/network/ifcfg-eth3
   BOOTPROTO='static'
   IPADDR='170.30.13.52/24'
   NAME='eth2'
   PREFIXLEN='24'
   STARTMODE='auto'
   USERCONTROL='no'
   LINK_REQUIRED='no'
   ```

   Not being up or having the wrong IP address could be caused by:

   - a missing Ethernet connection on the node due to a faulty cable
   - the cable not plugged into the proper network
   - an Ethernet switch with the incorrect VLAN

Boot Fails on a Node that Should be Disabled

Procedure

Did someone enable a node that had previously been disabled for some reason?

Look on the SMW in `/var/opt/cray/log/commands` for any `xtcli disable` and `xtcli enable` commands.

Boot Halts with an NMI when DEBUG Shell Entered

About this task

This procedure applies to CLE release 6.0.UP00 and 6.0.UP01 only.

If the boot of a node halts with a non-maskable interrupt (NMI) as soon as it drops into the DEBUG shell, the cause may be that the node is on a blade that has kernel parameter `stonith=true`.

With boot node failover and SDB node failover, the `stonith=true` parameter must be set on the blade that has the primary node. This is part of the required configuration so that the failover will happen from the primary node to the alternate node. However, this STONITH setting also applies to all of the other nodes on that blade.

When a system administrator wants to boot a node with the DEBUG shell to analyze a problem by stepping through the breakpoints in the `/init` script, they will set the kernel parameter DEBUG=true. Any node on a blade that has `stonith=true` and has the kernel parameter `DEBUG=true` will have an NMI error. It can also occur if the boot drops into the DEBUG shell for a different reason, such as a problem with Ansible code in the init phase. The node drops into the DEBUG shell and shows the DEBUG shell prompt, but before a command can be typed in the DEBUG shell, the NMI error message appears.

```
2016-06-16T15:59:36.173243-05:00 c0-0c0s9n1 DEBUG SHELL: pre configuration; exit
will resume
2016-06-16T15:59:36.173255-05:00 c0-0c0s9n1 DEBUG SHELL
2016-06-16T15:59:36.173486-05:00 c0-0c0s9n1 :/ # tsc: Refined TSC clocksource
calibration: 2099.986 MHz
2016-06-16T15:59:36.173644-05:00 c0-0c0s9n1 Switched to clocksource tsc
2016-06-16T16:02:09.065204-05:00 c0-0c0s9n1 Stop NMI detected on CPU 0
```

This issue has been fixed in CLE 6.0 UP.02. For sites running CLE 6.0.UP00 and UP01, try the following.

Procedure

1. Run the `xtdaemonconfig` command to determine whether the above node has `stonith=true` set for its blade.

   For example, to check for node c0-0c0s9n1, use the blade c0-0c0s9.

   ```
   smw# xtdaemonconfig c0-0c0s9 | grep stonith
   c0-0c0s9: stonith=true

   To disable this setting

   smw# xtdaemonconfig c0-0c0s9 stonith=false
   ```

```
Note: Setting stonith=false for this blade means that if the blade has the
primary boot node or primary SDB  node, then boot node failover or SDB node
failover will be unable to fail to the alternate node.
```

2. If `stonith=true` for the node, disable that setting.

```
smw# xtdaemonconfig c0-0c0s9 stonith=false
```

> **NOTE:** Setting `stonith=false` for this blade means that if the blade has the primary boot node or primary SDB node, then boot node failover or SDB node failover will be unable to fail to the alternate node.

Check Which Boot Automation File Being Used

Is the wrong boot automation file being used? The boot automation file passed to `xtbootsys` is logged to the beginning of the `/var/opt/cray/log/p0-current/bootinfo*` file.

```
smw# head /var/opt/cray/log/p0-current/bootinfo*
bootrecorder is available
NIMS is enabled
Using automation file '/opt/cray/hss/default/etc/auto.panda.start'
```

xtbootsys Fails with Undefined Partition

About this task

If `xtbootsys` fails with a message indicating that a partition is not defined, it may be that the partition parameter was omitted from the command.

Non-partitioned systems are assumed to be p0. If booting a partitioned system, the `--partition` parameter must be added to the `xtbootsys` command. For example,

```
smw# xtbootsys --partition p3
```

Possible Problem from Mismatch of Netroot Information on a Node

When Netroot is being used, the boot image should be `initrd-something` and the kernel parameter should be `netroot=something` for this node.

There are three ways to misconfigure this such that the node does not boot correctly:

- initrd boot image, but no Netroot kernel parameter
- non-initrd boot image, but has Netroot kernel parameter
- initrd boot image and Netroot kernel parameter, but the initrd boot image does not match the paired image root in the Netroot kernel parameter

Use this command to look for these situations:

```
smw# cnode list c2-3c0s5n2
```

Boot Fails on a Node using Netroot

About this task

When using Netroot for a login or compute node, the Netroot image root (the one that is listed in the node's Netroot kernel parameter) must be pushed from the SMW to the boot node before booting the login or compute node.

Procedure

1. Find the boot image for the node.

```
smw# cnode list | grep netroot
c0-0c0s1n0   service  login
/var/opt/cray/imps/boot_images/initrd-login-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio
    -           sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 NIMS_GROUP=login
netroot=login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160705
ids=10.128.0.37
config_set=p0
```

2. Check on the SMW for the image root that was shown in the Netroot kernel parameter in step 1.

```
smw# ls -l /var/opt/cray/imps/image_roots/\
login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160705
```

3. Check on the boot node for that image root.

```
boot# ls -l /var/opt/cray/imps/image_roots/\
login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160705
```

4. Take one of these actions, depending on whether the image root is on the boot node.

Option	Description
If image root missing from boot node	Push the image root from the SMW to the boot node. `smw# image push -d boot login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160705`

If image root on boot node	If the node failed to access the image root properly, perhaps the transfer of the image root was not complete before the node started to boot. In this case, warm boot the node.
	`smw# su crayadm`
	`crayadm@smw> xtbootsys --reboot c0-0c0s1n0`

If image root partially transferred to boot node	First, remove the image root directory from the boot node. **CAUTION:** Use this `rm` command on the **BOOT NODE ONLY**.

Option	Description

	`boot# rm -rf /var/opt/cray/imps/image_roots`
	
	Then, when the image root directory has been removed from the boot node, use the following command to push the image root from the SMW to the boot node again.
	
	`smw# image push -d boot \ login-large_cle_6.0.UP01-build6.0.96_sles_12-created20160705`

Diags Content Missing

About this task

If diags content is missing, was the diag image root pushed from the SMW to the boot node?

Procedure

1. Check on the SMW for the diags image root.

```
smw# ls -l /var/opt/cray/imps/image_roots/diag-all_cle_60up01_sles_12_x86-64
```

2. Check on the boot node for the diags image root.

```
boot# ls -l /var/opt/cray/imps/image_roots/diag-all_cle_60up01_sles_12_x86-64
```

3. Take one of these actions, depending on whether the image root is on the boot node.

Option	Description
If image root missing from boot node	Push the image root from the SMW to the boot node. `smw# image push -d boot diag-all_cle_60up01_sles_12_x86-64`

If image root on boot node	If the node failed to access the image root properly, perhaps the transfer of the image root was not complete before the node started to boot. In this case, warm boot the node.
	
	`smw# su crayadm`
	`crayadm@smw> xtbootsys --reboot c0-0c0s1n0`

If image root partially transferred to boot node	Remove the image roots directory and push the image root from the SMW to the boot node again.
	
	`boot# rm /var/opt/cray/imps/image_roots/diag-all_cle_60up01_sles_12_x86-64`
	
	`smw# image push -d boot diag-all_cle_60up01_sles_12_x86-64`

PE Software Content Missing

About this task

If Cray Programming Environment (PE) software content is missing, was the PE software image root pushed from the SMW to the boot node?

Procedure

1. Check on the SMW for the PE software image root.

```
smw# ls -l /var/opt/cray/imps/image_roots/pe_compute_cle_6.0up01_sles_12
```

2. Check on the boot node for the PE software image root.

```
boot# ls -l /var/opt/cray/imps/image_roots/pe_compute_cle_6.0up01_sles_12
```

3. Take one of these actions, depending on whether the image root is on the boot node.

Option	Description
If image root missing from boot node	Push the image root from the SMW to the boot node. `smw# image push -d boot pe_compute_cle_6.0up01_sles_12`

If image root on boot node	If the node failed to access the image root properly, perhaps the transfer of the image root was not complete before the node started to boot. In this case, warm boot the node.
	`smw# su crayadm`
	`crayadm@smw> xtbootsys --reboot c0-0c0s1n0`

If image root partially transferred to boot node	Remove the image roots directory and push the image root from the SMW to the boot node again.
	`boot# rm /var/opt/cray/imps/image_roots/pe_compute_cle_6.0up01_sles_12`
	`smw# image push -d boot pe_compute_cle_6.0up01_sles_12`

Node Fails to Mount Config Set from IDS Servers

About this task

If a node fails to mount the config set from IDS servers, find out if incorrect or unreachable nodes are set in that node's IDS kernel parameter.

Procedure

Check the IDS kernel parameter for the node.

```
smw# cnode list c1-2c0s5n3
NAME         TYPE     GROUP    IMAGE
                                        CONFIG_SET  EXT_PARAMETERS

c0-0c0s11n3  compute  compute
/var/opt/cray/imps/boot_images/initrd-compute-large_cle_6.0.UP01-
build6.0.96_sles_12-created20160705.cpio  -
    NIMS_GROUP=compute netroot=compute-large_cle_6.0.UP01-build6.0.96_sles_12-
created20160705
ids=10.128.2.70,10.128.1.201,10.128.0.195 config_set=p0
```

This example shows `ids=10.128.2.70,10.128.1.201,10.128.0.195`. Check which nodes are assigned these three IP addresses. The node in this example is dependent on these three nodes to get config set information from IDS.

Did any of these nodes have a problem booting? If all of them had a problem booting, then that is why this node had a problem. With Cray scalable services, tier3 nodes (most of the CLE nodes) all have tier2 nodes listed in their IDS kernel parameter. And tier2 nodes all have tier1 nodes listed in their IDS kernel parameter. The tier1 nodes all have the SMW eth3 IP address (10.3.1.1) in their IDS kernel parameter. With SMW HA, the tier1 nodes still have 10.3.1.1 in the IDS kernel parameter, because that IP address is used by the first SMW in the SMW HA pair.

Put a Node in DEBUG and Step Through the Init Phase

Prerequisites

This procedure requires two windows to be open:

- window 1 for connecting to the node in question (c8-0c2s8n1/nid01697 in this example) using `xtcon`
- window 2 for typing administrative commands on the SMW

About this task

The `/init` script runs during the early stages of a boot, and it contains breakpoints that enable a user to examine various system values and files during the boot. Check `/init` on the SMW in `/var/opt/cray/imps/image_roots/`*`node_image_root`*, where *`node_image_root`* is the image root directory corresponding to the boot image for the node.

Here are some of the breakpoints in `/init`. The list of breakpoints in the `/init` script may change over time and software releases.

```
${DEBUG} && echo "DEBUG SHELL: in setup_netroot; exit will init vars" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: in setup_netroot; exit will contruct netroot" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to DVS lower mount; exit will resume" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to chroot prep; exit will resume" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: post netroot preload debug; exit will resume" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to mounting merge layer tmpfs" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to chroot Ansible; exit will resume" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to bind mounts of upper and lower; exit will
```

```
resume" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to switch_root; exit will switch" > $
{con_debug}
${DEBUG} && echo "DEBUG SHELL: pre configuration; exit will resume" > ${con_debug}
${DEBUG} && msg_severity=3 && echo "DEBUG SHELL: pre config service; exit will
resume" > ${con_debug}
${DEBUG} && msg_severity=3 && echo "DEBUG SHELL: pre ansible/netroot; exit will
resume" > ${con_debug}
${DEBUG} && msg_severity=3 && echo "DEBUG SHELL: pre systemd; exit will resume" > $
{con_debug}
```

Procedure

1. In window 1, connect to the node and wait to interact with the DEBUG shell (occurs in step 5).

```
smw# xtcon c8-0c2s8n1
```

2. In window 2, check the node's current parameters.

```
smw# cnode list c8-0c2s8n1
NAME        TYPE     GROUP    IMAGE

CONFIG_SET       EXT_PARAMETERS
c8-0c2s8n1  compute  compute
/var/opt/cray/imps/boot_images/roe-initrd-compute-large_cle_6.1.DV00-
build6.1.50DV_sles_12-created20160201.cpio
  p0-brt-20160202  hsn_ipv4_mask=255.252.0.0 hsn_ipv4_net=10.128.0.0
sdbnodeip=10.131.255.253
bootnodeip=10.131.255.254 NIMS_GROUP=compute
netroot=compute-large_cle_6.1.DV00-build6.1.50DV_sles_12-created20160201
ids=10.128.8.216,10.128.0.66,10.128.3.202 config_set=p0-brt-20160202
```

3. In window 2, update the node's parameter so that `/init` will drop into the DEBUG shell.

```
smw# cnode update -k DEBUG=true c8-0c2s8n1
smw# cnode list c8-0c2s8n1
NAME        TYPE     GROUP    IMAGE

CONFIG_SET       EXT_PARAMETERS
c8-0c2s8n1  compute  compute
/var/opt/cray/imps/boot_images/roe-initrd-compute-large_cle_6.1.DV00-
build6.1.50DV_sles_12-created20160201.cpio
  p0-brt-20160202  sdbnodeip=10.131.255.253 bootnodeip=10.131.255.254
hsn_ipv4_mask=255.252.0.0
hsn_ipv4_net=10.128.0.0 DEBUG=true NIMS_GROUP=compute
netroot=compute-large_cle_6.1.DV00-build6.1.50DV_sles_12-created20160201
ids=10.128.8.216,10.128.0.66,10.128.3.202 config_set=p0-brt-20160202
```

4. In window 2, reboot the node to use the new kernel parameters.

```
crayadm@smw> xtbootsys --reboot -r "testing init" c8-0c2s8n1
```

5. In window 1, interact with the DEBUG shell.

   NOTE: While in the DEBUG shell, type "exit" in the console terminal to advance `/init` to the next breakpoint.

6. In window 2, when done interacting with the DEBUG shell, remove the DEBUG kernel parameter from this node and confirm its removal.

```
smw# cnode update -K DEBUG c8-0c2s8n1
smw# cnode list c8-0c2s8n1
```

At this point, the system administrator may wish to keep the node running as it is or warm boot the node to ensure that the next boot will come up cleanly and not in DEBUG mode.

Information to Gather for Opening a Bug

If a bug needs to be opened, collect the information from `xtshowrev`, `xtdumpsys` and `cdump`, `xtcheckhss`, any pertinent log files, and all or a portion of the global config set and CLE config set. The following sections show how to gather this information.

Collecting information about an SMW HA system entails using several commands in addition to those used for a non-HA system. The final section describes these commands and provides examples.

xtshowrev

Every bug opened should include information about the XC system and the software installed. This information is displayed by the `xtshowrev` command on the SMW. For command usage, type `xtshowrev -h`.

```
crayadm@smw> module load xtshowrev
crayadm@smw> xtshowrev
Site:                CRAY/INTERNAL
S/N:                 9999
System Type:         XC40
Install Date:        2016-06-15
System Name:         panda
CNL/CLE Release:     6.0.UP01
XT Release:          6.0.96
CLE Kernel:          3.12.51-52.31.1_1.0600.9146
CLE OS:              SLES12
CLE Patch Sets:      02 03 04 06 07
CLE FNs:
Lustre Version:      2.7
OS Type:             CLE
SMW Release:         8.0.UP01
SMW Build:           8.0.96
HSS Release:         8.0__446__ge75851a-49.1
SMW Kernel:          3.12.51-52.39
SMW OS:              SLES12
SLE Patch Sets:
SMW Patch Sets:      02
SMW FNs:             5844c
SEC Release:         Cray_SEC 8.0__6__g689802a (sec 2.7.6)
Current Date:        2016-07-06 14:59:10
crayadm@smw>
```

xtdumpsys and cdump

For most bugs, it is helpful to include the output from `xtdumpsys` and `cdump`.

The `xtdumpsys` program dumps logs and other information from a running system into a directory in `/var/opt/cray/log`. This is very useful in the case of a node crash or failure to boot. `xtdumpsys` is often used in combination with `cdump` of a specific node or nodes that may have crashed or hung.

`xtdumpsys` collects and analyzes information from a Cray XC system that is failing or has failed, has crashed, or is hung. If doing a full dump (recommended), `xtdumpsys` will gather the following by default:

- event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors
- config sets from SMW
- NIMS logs from SMW
- Ansible logs from nodes (only those nodes that have Ansible failures in the console log)
- Ansible changed-files log from nodes

  NOTE: `xtdumpsys` does not gather the Ansible changed-files log unless the `--add` option is used to specify which nodes to gather it from.

The `cdump` command is used to dump node memory to a file from a panicked or hung node. It is typically used with the `-A` option to automatically include important supporting data files, such as `vmlinux` and `systemp.map`.

To use `xtdumpsys` and `cdump` together, open a second terminal window. In window 1, enter one of the following.

- For a full dump:

```
window1 crayadm@smw> xtdumpsys [--partition pN] -r 'REASON'
```

- For a full dump that includes Ansible changed-files log from nodes in *`NODE_LIST`* (a space-separated list of nodes):

```
window1 crayadm@smw> xtdumpsys [--partition pN] -r 'REASON' \
--add NODE_LIST
```

- For a partial dump of only the NIMS and Ansible information from nodes in *`NODE_LIST`* (a space-separated list of nodes):

```
window1 crayadm@smw> xtdumpsys [--partition pN] -r 'REASON' \
--add NODE_LIST --plugins-include ansible_logs ansible_changed_files \
lsb_path_logs nims_logs
```

  Note that the LSB Path Logs plugin is needed in this command because it collects the console log, which is used to determine which nodes have Ansible failures.

Then in window 2, do the following:

1. For hung CLE node(s), use `xtnmi` to kill the node(s) before capturing the cdump. Note that the *`ID_LIST`* is a comma-separated list of nodes.

```
window2 crayadm@smw> xtnmi -k halt ID_LIST
```

2. Use the cdump command to capture information about the node(s). Note that *`NODE_LIST`* is a space-separated list of nodes.

```
window2 crayadm@smw> cd <dumpdir>
window2 crayadm@smw> mkdir cdump
window2 crayadm@smw> cd cdump
window2 crayadm@smw> cdump -A -r xt-hsn@boot-pN NODE_LIST
```

If a `cdump` has been taken, it can be examined using the `crash` command on the SMW.

For more information about these commands, see their man pages.

xtcheckhss

A few bugs may require the output from `xtcheckhss`.

The `xtcheckhss` command is designed to validate the health of the HSS by gathering and displaying information supplied by scripts located on blade controllers (BC) and cabinet controllers (CC) on a Cray XC Series system. `xtcheckhss` includes these tests:

Version Checker Reads the current firmware version running on the L0C, QLOC, L0Ds, BC micro, CC micro, CC FPGA, CHIA FPGAs, Tolapai BIOSes, and Node BIOS. The version that is read from each device is compared to the currently installed versions on the SMW.

Sensor Checker Reads environment sensors including temperatures, voltages, currents, and other data.

SEEP Checker Reads serial electrically erasable PROMs (SEEP) in the system. This test can report any un-initialized, zeroed, or unreadable SEEPs.

AOC Checker Reads all active optical cable (AOC) data. This test displays any outliers relative to the average data calculated by previous runs.

ITP Checker Validates the embedded in-target probe (ITP) path. A simple power check is performed via the ITP.

NTP Checker Reads system time on all controllers and compares them with the SMW time; displays any mismatches.

Control Checker Examines system controls.

- Performs a Micronet throughput test on the CC or BC microcontroller.
- Gets the uptime and reset cause of the CC or BC microcontroller.
- Verifies that the rectifiers are outputting the correct voltage for their current state.
- Verifies that the Aries sensors are reading expected values for its current state.
- Verifies that the blower sensors are reading expected values for its current state.
- Prints the blade type and the voltage regulator module (VRM) type.
- Validates that water valves are open and that sensors are within expected range.

Configuration Information Checker Reads the system hardware configuration and reports the system setup, including the blade type, daughter card type, CPU type and count, and the CPU and processor daughter card (PDC) mask.

PCI checker Checks for missing or degraded PCIe (peripheral component interconnect express) connectivity on add-in cards on an I/O base blade (IBB). This test requires that the nodes be powered up and bounced. Any cards that do not train to the PCIe Gen or Width specified in the Link Capability register are flagged. Any cards that are reported as physically present but not seen by the node are flagged.

NOTE: If any of these show a warning or error, further investigation may be needed.

```
crayadm@smw> xtcheckhss
No Version Mismatches Found!
No Sensor Warnings Found!
No SEEP Errors Found!
No AOC Errors Found!
No ITP Errors Found!
No NTP Time Sync Errors Found!
```

```
No Control Errors Found!
No Info Errors Found!
No Expander Errors Found!
No PCIe Card Errors Found!
crayadm@smw>
```

log files

Although some SMW and Ansible logs (from a node that had an error when running cray-ansible) may be included in `xtdumpsys`, other SMW and Ansible logs may be useful in debugging a boot failure. See *SMW Log File Locations* on page 19 for specific log files and which daemons or processes write to them.

Further requests for information may include running commands to capture their output.

Collect Additional Information from SMW HA Systems

Because `xtshowrev` does not yet collect software release info for SMW HA systems, use this command to gather that information:

```
smw# cat /etc/opt/cray/release/smwha-release
RELEASE=12.0.UP00
BUILD=12.0.48
DATE=201605180109
ARCH=x86_64
```

To gather information about the SMW HA configuration, use this command (this example shows the output for an HA pair "minnie" and "mickey"):

```
smw# /opt/cray/ha-smw/default/sbin/ha_health

Cluster State
-----------------------------------------
Health State                : Healthy
Active Node                 : minnie
Node-1                      : mickey (online)
Node-2                      : minnie (online)
Number of Resources         : 33
Number of Resources Running : 33
Number of Resources Stopped : 0
Maintenance Mode            : disabled
Stonith Mode                : enabled
```

To gather information about the distributed replicated block device (DRBD), use this command:

```
smw# cat /proc/drbd
version: 8.4.4 (api:1/proto:86-101)
GIT-hash: 3c1f46cb19993f98b22fdf7e18958c21ad75176d build by SuSE Build Service
 0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
  ns:905180 nr:0 dw:905772 dr:12762 al:51 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

To check the cluster status, use this command on either of the SMWs (this example shows the output for an HA pair "minnie" and "mickey"):

```
smw# crm_mon -1r
Last updated: Tue Sep  6 11:26:06 2016
```

```
Last change: Tue Sep  6 08:52:13 2016
Stack: corosync
Current DC: minnie (167903490) - partition with quorum
Version: 1.1.12-ad083a8
2 Nodes configured
33 Resources configured

Online: [ mickey minnie ]

Full list of resources:

 ClusterIP      (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP1     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP2     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP3     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterIP4     (ocf::heartbeat:IPaddr2):       Started minnie
 ClusterMonitor (ocf::smw:ClusterMonitor):      Started minnie
 ClusterTimeSync        (ocf::smw:ClusterTimeSync):     Started minnie
 HSSDaemonMonitor       (ocf::smw:HSSDaemonMonitor):    Started minnie
 Notification   (ocf::heartbeat:MailTo):        Started minnie
 ResourceInit   (ocf::smw:ResourceInit):        Started minnie
 cray-cfgset-cache      (systemd:cray-cfgset-cache):    Started minnie
 dhcpd  (systemd:dhcpd.service):        Started minnie
 fsync  (ocf::smw:fsync):       Started minnie
 hss-daemons    (lsb:rsms):     Started minnie
 stonith-1      (stonith:external/ipmi):        Started mickey
 stonith-2      (stonith:external/ipmi):        Started minnie
 Resource Group: HSSGroup
     postgresqld        (lsb:postgresql):       Started minnie
     mysqld     (ocf::heartbeat:mysql): Started minnie
 Resource Group: IMPSGroup
     cray-ids-service   (systemd:cray-ids-service):     Started minnie
     cray-ansible       (systemd:cray-ansible): Started minnie
     IMPSFilesystemConfig       (ocf::smw:FileSystemConfig):    Started minnie
 Resource Group: LogGroup
     rsyslog    (systemd:rsyslog.service):      Started minnie
     cray-syslog        (systemd:llmrd.service):        Started minnie
     LogFilesystemConfig        (ocf::smw:FileSystemConfig):    Started minnie
 Resource Group: SharedFilesystemGroup
     homedir    (ocf::heartbeat:Filesystem):    Started minnie
     md-fs      (ocf::heartbeat:Filesystem):    Started minnie
     imps-fs    (ocf::heartbeat:Filesystem):    Started minnie
     ml-fs      (ocf::heartbeat:Filesystem):    Started minnie
     repos-fs   (ocf::heartbeat:Filesystem):    Started minnie
     pm-fs      (ocf::heartbeat:Filesystem):    Started minnie
 ip-drbd-pgsql  (ocf::heartbeat:IPaddr2):       Started minnie
 Master/Slave Set: ms-drbd-pgsql [drbd-pgsql]
     Masters: [ minnie ]
     Slaves: [ mickey ]
```

Note that the `crm_mon` output displays the SMW host names as "Online" in alphanumeric order; the first SMW shown is not necessarily the primary SMW.