System Administration for Cray XE and XK Systems

Rick Slick
Sr. Technical Instructor
Cray Inc.

Abstract

The Cray Linux Environment requires tasks and processes beyond what is required for managing basic Linux systems. This short seminar covers some system administration basics, as well as a collection of tools and procedures to enhance monitoring and logging and efficient command usage. The talk will include new capabilities in logging, Node Health Check, and ALPS. New features in recent releases will also be discussed. The session is geared towards new system administrators, as well as those with more experience.

Topics

- Lightweight Log Manager (LLM)
- Simple Event Correlator (SEC)
- Parallel Command (`pcmd`)
- ALPS `aprun` Epilog and Prolog
  - Other new ALPS features
- Node Health Checker (NHC)
- Programming Environment (PE) Installer

Lightweight Log Management (LLM)

- **The Cray Lightweight Log Management (LLM) system is the log infrastructure for Cray systems going forward**
  - At a high level, a library is used to deliver messages to `rsyslog` utilizing the RFC 5424 protocol, and `rsyslog` transports those messages to the SMW and places the messages into log files
    - Benefits of LLM
      - Leverages open-source, third-party software
      - Standards based
      - Light weight
      - Provides a standard logging interface for Cray software
      - Logs kept in a central location
      - More consistent messages
      - Flexibility
      - Log-forwarding capability

LLM

- **High-level Message Flow**
  - `rsyslog` on all Cray system service nodes forwards messages to the boot node (or optionally defined dedicated Syslog node)
  - The boot node (or any other service node that can reach the SMW) forwards its messages directly to the SMW.
  - All messages in the LLM system are forwarded to the SMW and are not kept locally on the service nodes
    - The service nodes do not make use of persistent storage, except for temporary files in a failure situation

LLM

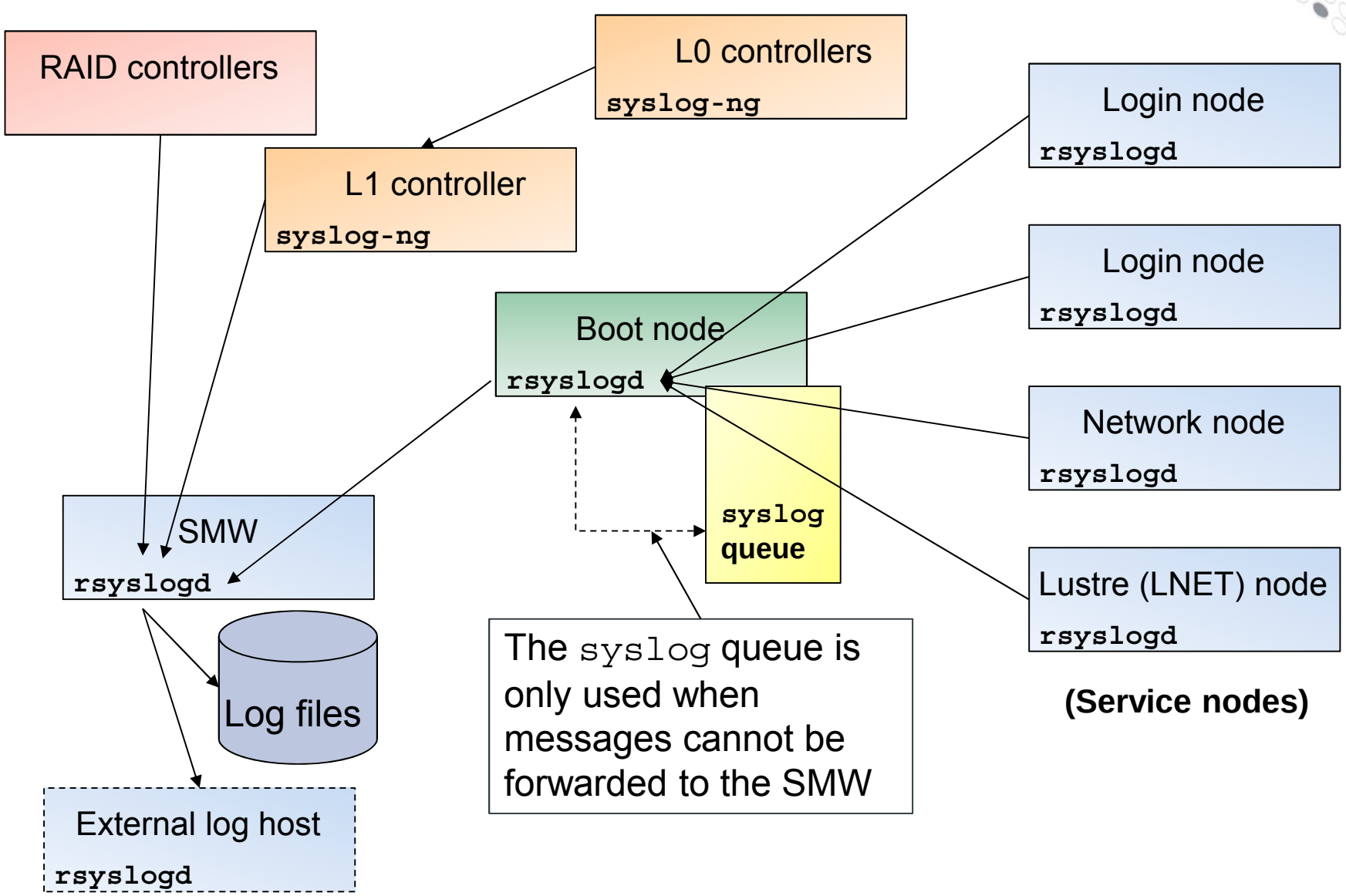
- **Getting Messages to `rsyslog`**
  - Although the RFC 5424 is a standard `syslog` format, currently the standard methods to send messages to `syslog`, such as `syslog()`, do not support the RFC 5424 format
    - As a result, LLM provides a library to simplify and standardize logging for Cray applications
    - Applications log messages to this library, which formats the message in RFC 5424 according to LLM conventions and then delivers the message to `rsyslog` on the local machine
  - On forward-only hosts, such as the login node, outgoing messages are set so that the hostname appears in **cname** format
  - Additionally the L1 and L0 log files are forwarded to:

```
/var/opt/cray/log/controller/<cabinet>/<controller>/messages-<yyyymmdd>
/var/opt/cray/log/controller/<cabinet>/<controller>/coldstart-<node>-
<yyyymmdd>
```

LLM

- **Production ready at SMW 7.0 and CLE 4.1**
  - Make sure `rsyslogd` and the Lightweight Log Manager recovery daemon (`llmrd`) are running after upgrades
  - On the SMW, Boot node and in shared root:
    - Review `/etc/opt/cray/llm/xttrim.conf`
    - Review `/etc/opt/cray/llm/llm.conf`
      - If forwarding logs from a RAID controller ensure that the `llm_raid_ip` is set correctly
    - On the SMW verify `/var/opt/cray/log` is on dedicated hard drive
  - On the SMW in `CLEinstall.conf`
    - The `alps_logMethod` should be set to 1 for traditional log files, 2 for LLM logging, or 3 (default) for both traditional log files and LLM logging
      - This will be reflected in `/etc/alps.conf` on the system
        - `alps_logMethod=3`

LLM Diagram


RAID controllers
L0 controllers
syslog-ng
L1 controller
syslog-ng
Login node
rsyslogd
Login node
rsyslogd
Boot node
rsyslogd
Network node
rsyslogd
syslog
queue
SMW
rsyslogd
Lustre (LNET) node
rsyslogd
Log files
The syslog queue is only used when messages cannot be forwarded to the SMW
(Service nodes)
External log host
rsyslogd

Log File Locations

- **SMW Log messages**
  - `/var/opt/cray/log/smwmessages-<date>`
- **Event Router Daemon (ERD)**
  - `/var/opt/cray/log/event-<date>`
- **RAID controller messages**
  - `/var/opt/cray/log/raid-<date>`
- **ALPS logs**
  - `/var/opt/cray/log/p0-current/alps/<process>-<date>`
    - Where `p0-current` is a link to the current boot session directory
    - Where process is: `apbridge, apevent, apres, aprun, apsched, apsys, or apwatch`
- **Console logs for larger systems, LLM can be configured to log to a `console` directory in `p0-current` and rotate the logs hourly**

Log File Locations

- **L1 and L0 controller**
  - /var/opt/cray/log/controller/<cabinet>/<controller>
  - Where `<cabinet>` is the `cname` of the cabinet
  - Where `<controller>` is the `cname` of the L1 or L0 controller

Additional LLM Information

- **On the SMW to set your environment to find all the man pages: `module load llm-utils`**
  - Man pages:
    - `intro_LLM`
    - `intro_LLM_logfiles`
    - `llm.conf`
    - `xttrim.conf`
    - `cray-syslog`
    - `llmrd`

SEC (Simple Event Correlator)

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

SEC

- **SEC has been installed successfully on many Cray systems**
- **A set of SEC software and rules (configuration files) are located on an internal file system**
  - Cray personnel download and install the package on the SMW
  - Installation is quick and simple
    - Required information is *system name* and *serial number*
    - Additionally an email alias can be created for distribution of messages, default is to send email notification to crayadm
  - Once configured add an entry to the auto script on the SMW start SEC automatically when the system boots
  - SEC will terminate automatically on system shutdown

SEC

- **SEC supports:**
  - Both LLM (Lightweight Log Management) and non-LLM logging and is automatically determined, with no changes needed to switch between the two
    - The same rule-set supports both
      - No changes were needed to the rules to support LLM, the changes needed were in the start-up and other scripts
  - SEC automatically detects if LLM is performing hourly or daily console log rotation and determines the location of console logs
    - LLM provides hourly console log rotation for large mainframes that can at times produce extremely large daily console log files that are too large to peruse or `grep`
  - There are rules for GPUs that are used when GPUs are present
    - Currently a configuration parameter, later auto-detection of GPUs
  - Optional automatic submittal of SFDC cases for appropriate memory failures
    - This can be toggled on/off via a variable in SEC_VARIABLES

SEC Management

- The SEC files are typically installed on the SMW in:
  `/home/crayadm/sec`
- To start SEC manually:
  `~/sec/bin/start_sec.sh`
  - Starting SEC will terminate and previous instances of SEC
- To stop SEC manually:
  `~/sec/bin/stop_sec.sh`
- To restart SEC:
  `~/sec/bin/restart_sec.sh`
- In older releases these files were in
  `~sec/distributed_rules/`

SEC Directories and Files

- **Three directories in the distribution are:**
  - `~/sec/rules/basic`
    - Contains rules used by SEC on a Cray system
    - In older releases the rules were in `~/sec/distrubuted_rules`
  - `~/sec/rules/local`
    - Place to store local rules created to use with SEC
    - Using this allows an update of the distributed rules without affecting local rules
    - If you rewrite a *distributed* rule it is suggested you disable the original rule and copy the file to the *local* directory
      - You disable a rule by changing the suffix of the rule from *.sr to something like *.sr_copied_to_local
    - In older releases this was `~/sec/local_rules`
  - `~/sec/bin`
    - Contains commands and scripts used by SEC
    - In older releases these were in `~/sec/distrubuted_rules`

SEC_VARIABLE File

- A partial SEC_VARIABLE file is shown here:

```
type= Single
ptype= RegExp
pattern= (SEC_STARTUP|SEC_RESTART|SEC_SOFTRESTART)
context= [SEC_INTERNAL_EVENT]
desc= $0
action= eval %N (return (" \n");) ;\
        assign %host Cray-lake ;\
        assign %SNUM 9000 ;\
        assign %SN sn%{SNUM} ;\
        assign %list crayadm ;\
        assign %sfdc_submittal_master disabled ;\
        assign %sfdc_mce_disable_node off ;\
        assign %sfdc_mce_L0 disabled ;\
        assign %sfdc_mce_threshold disabled ;\
        assign %sfdc_mce disabled ;\
        assign %sfdc_dbe_gpu disabled ;\
        assign %gpu_rules disabled ;\
        assign %rules_dir /home/crayadm/sec/rules ;\
        assign %sec_log_dir /var/log/sec ;\
```

Additional Information on SEC

- A brief introduction to SEC is found at:
  - http://arstechnica.com/information-technology/2005/05/linux-20050519/
    - Look under "Tools, Tips, and Tweaks“
- The man page and FAQ are helpful, as are the part 1 and part 2 tutorials at this Web site
  - http://simple-evcorr.sourceforge.net
- Good information can also be searched for from the newsgroup:
  - http://sourceforge.net/search/?group_id=42089&type_of_search=mlists

check_xt.sh

- **There are actually two files associated with this, `check_xt.sh` and `check_xt_wrapper.ex`**
  - `check_xt_wrapper.ex`
    - Is a wrapper script and is called via *crayadm's* `crontab`, it calls `check_xt.sh` and provides a timeout capability for calls to `ssh` made within `check_xt.sh`
      - `ssh` itself lacks it's own timeout capability, so this script provides it
  - `check_xt.sh`
    - This script is run on the SMW and reports on state changes of the XE/XK/XT system and logs data about the state of nodes and jobs
      - For example: compute nodes going down or system boots

Congestion Tools

- **Congestion protection event information is present in the SMW `xtnlrd` log file, but it can be difficult to parse through**
- **`xtcpreport` and `xtcptop` provide a simple way to view congestion protection events using a `xtnlrd` log file**
- **`xtcpreport` parses the given `xtnlrd` log file and presents information about congestion protection events to the user**
  - `xtcpreport` displays the time and duration of each congestion protection event, along with the `apid` of the application and other associated information
- **`xtcptop` is the real time version of `xtcpreport`**
  - Given a `xtnlrd` log file that is being updated by a running system, `xtcptop` will show whether or not there is a current congestion protection event, and if there is `xtcptop` will show the time, duration, `apid`, and other information associated with that event

Parallel Command (pcmd)

- **The `pcmd` is a tool that executes the same commands on compute nodes in parallel**
  - Similar to pdsh
  - All node output is displayed on the node where `pcmd` was launched
  - The summary option collects the identical responses into one output line
  - Specify nodes to run on either on the command line or in a file
  - Alternately, specify the APID and run on the nodes belonging to that APID
    - Automatically finds your APID if you are only running one job
- **Cautions:**
  - Can cause jitter
  - Obviously, don't delete files or processes that your application is using

pcmd Security

- **Uses Secured Sockets Layer (SSL)**
  - Users can only run a command on nodes where they have a job running
  - Root user can run on any and all nodes
  - `pcmd` is installed as a root-only tool
  - Must be installed as `setuid` root for unprivileged users to use it

Additional Information for `pcmd`

- **The pcmd runs in the DSL root; users can access commands in the shared root.**
  - Users can set the ENVIRONMENT VARIABLE: PCMD_APID
  - The RETURN VALUE code is 0 on success and 1 on failure
  - The `oom_score_adj` is set so that it is killed first
  - `pcmd` attaches the command to the APID
    - This means it is killed if the app is killed.
  - The command is run it from a login node
    - If it is not in your default path, then you should load the `nodehealth` module:
      `module load nodehealth`

pcmd Examples

- Without the summary option:

```
# pcmd -n 1,2,3,4 "./even_odd.sh"
Output from node nid00001:
Odd
Reply (complete) from nid00001 exit code: 0
Output from node nid00002:
Even
Reply (complete) from nid00002 exit code: 0
Output from node nid00003:
Odd
Reply (complete) from nid00003 exit code: 0
Output from node nid00004:
Even
Reply (complete) from nid00004 exit code: 0
```

- With the summary option:

```
#pcmd -s -n 1,2,3,4 "./even_odd.sh"
Output from 1,3:
Odd
Reply (complete) from 1,3 exit code: 0
Output from 2,4:
Even
Reply (complete) from 2,4 exit code: 0
```

ALPS `aprun` Prolog and Epilog

- **Allows administrators to define custom actions to be performed before and after a user's binary executes**
  - The use of the Prolog and Epilog are optional
    - They are set up in the ALPS configuration file (alps.conf)
    - Are started by the `apsys` agent of the `aprun`
    - Run as root on the service node
    - Provide *read-only* access (via environment variables) to specific values
  - **Note: The use of this feature removes any assurance of launch speeds**

Prolog/Epilog Setup

- Additions to `alps.conf`

```
apsys debug 1
        prologPath /usr/local/adm/sbin/prolog
        epilogPath /usr/local/adm/sbin/epilog
        prologTimeout 10
        epilogTimeout 10
/apsys
```

- If *prolog* or *epilog* file is not owned by root the `/var/log/alps/apsys` file will reflect the error

```
2012-11-24 10:41:24: configure:392: Error: prolog file is not owned by root.
2012-11-24 10:41:24: configure:431: Error: epilog file is not owned by root.
2012-11-24 10:41:24: Prolog and epilog will not be used because of error.
```

Log File Details

- In the log file: `/var/log/alps/apsys`

```
2012-11-21 15:29:23: *** apsys pid 7497 opening log file /var/log/alps/apsys
20121121 2012-11-21 15:29:23: *** apsys build: Nov 18 2012 23:38:19, rev
2012-11-21 15:29:23: *** apsys log method(s): Traditional LLM
2012-11-21 15:29:23: *** apsys time zone: UTC-0600 2012-11-21 15:29:23: Read
configuration file.
2012-11-21 15:29:23: Debug level set to 0x1
2012-11-21 15:29:23: Subsequent agents will use [/usr/local/adm/sbin/prolog] as
prolog with timeout of 10 seconds
2012-11-21 15:29:23: Subsequent agents will use [/usr/local/adm/sbin/epilog] as
epilog with timeout of 10 seconds
2012-11-21 15:29:57: [7516] Agent for apid 744934 prolog:
/usr/local/adm/sbin/prolog ...
2012-11-21 15:30:01: [7511] Agent for apid 744934 prolog done (0)
2012-11-21 15:30:08: [7669] Agent for apid 744934 epilog:
/usr/local/adm/sbin/epilog ...
2012-11-21 15:30:12: [7511] Agent for apid 744934 epilog done (0)
```

ALPS `aprun` Prolog/Epilog

- **A Few More Details:**
  - If the prolog fails, the aprun command will not execute
  - The epilog runs before the node health check
  - If the aprun uses MPMD, the ALPS_PREP_WIDTH variable (one of the read-only environmental variables) will reflect that
    - For example:

      ```
      aprun -n 400 ./cmd1 : -n 200 ./cmd2
      ```

    - Will results in: `ALPS_PREP_WIDTH=400:200`

Additional ALPS Command Options

- `aprun` compute unit affinity
- `aprun` reconnect and re-launch
- `apstat` display output format

Compute Unit Affinity

- The AMD Interlagos processors are composed of multiple compute units (Bulldozer modules)
  - A “Compute Unit” (CU) is a Cray term which represents a collection CPUs that share compute resources
  - Each Compute unit contains:
    - Two integer cores
    - Two 128-bit floating point units which can be combined into one shared, 256-bit
    - A shared floating point unit
    - A shared L2 cache
  - Due to the shared resources some codes may perform better using only one core per compute unit
  - Compute Unit Affinity provides a way to control placement of PEs to utilize or avoid these shared resources

Compute Unit Affinity

- **Across compute nodes, the `aprun -j` option controls the number of CPUs available per CU**
  - `-j 0` means to use all CPUs on each CU, the default is `-j 1`
    - `-j num_cpus` keeps the first num_cpus of each CU which are within the *cpuset* and removes the rest from consideration for application placement
    - `-cc cpu_list` overrides `-j` and most other placement options, within a compute node
  - Placement option processing priority within a compute node:
    - The `-sl` and `-sn` options create *cpusets* which force the rest of the placement options to be computed with respect to these *cpusets*
    - The `-N`, `-d`, and `-S` all operate within the set of remaining CPUs
    - The `-r cores` option removes selected CPUs from consideration for application placement and dedicates them for core specialization (corespec)

aprun Reconnect/Relaunch

- **Infrastructure support for application resiliency**
  - No batch system involvement
  - User requested through aprun options
  - Node failure events, not general software errors
    - Relaunch application following node failures
    - Reconnect application fanout tree following node failures or connect failures
- **User must explicitly ask for *relaunch* capability**
  - `aprun -R max-reduced-PEs`
    - "`aprun -R 64`" relaunch with up to 64 fewer PEs
    - "`aprun -R 0`" relaunch with same number of PEs
  - User can reserve additional spare nodes at batch submission(qsub), if desired
    - There is **NO** system level pool of spare nodes
    - Can't be used with MPMD mode

Application Relaunch

- **Application is killed on compute nodes before a relaunch is attempted; node health is invoked**
- **Only `aprun` knows about a *relaunch***
  - *Relaunch* looks like new launch to `apsched`
  - New `apid` assigned, new placement list from the same original batch reservation
  - Must be sufficient nodes reserved within the reservation to support *relaunch* of application using `aprun -R` value of maximum fewer PEs
- **No application changes required**
  - Typical an application using this feature will already have its own checkpoint/restart capability
  - `aprun` relaunches the application with environment variable `ALPS_APP_RELAUNCH` set to the `relaunch` attempt number
  - The application decides whether to continue executing from beginning or access a checkpoint file

Application Reconnect

- **ALPS provides infrastructure support**
  - PMI, programming model and application changes required to use this feature
    - Due to programming model and application required changes, very limited use of this feature for the foreseeable future
  - User must explicitly ask for reconnect capability: `aprun -C`
  - Reconnect fanout control tree around failed nodes
    - Failed nodes are not replaced regardless of how many nodes are reserved within the batch reservation
  - `aprun -R` and `-C` are mutually exclusive

Application Reconnect

- **`aprun` and `apinit` know that user has requested reconnect following node failure or connect failure**
  - Following node or connect failure the application is not killed
  - The `apid` remains the same and `apsched` is not involved
  - Node health is not invoked prior to reconnect Activity
- **`aprun` reads reservations file in case more nodes have failed and sends that failed list to PE0 (head node)**
  - As needed, `aprun` will select a new PE0 (head node) if the head node fails
  - Each `apinit` recalculates its controlled child nodes and contacts new controlled nodes and releases prior child nodes, as needed
  - The fanout control tree is recreated around failed nodes
- **Once `aprun` has been notified that the fanout control tree has been successfully reconnected, the list of failed nodes is provided to PMI on each remaining compute node for that application**
  - Any additional node failures during a reconnect activity result in another reconnect activity starting
  - No limit on number of compute nodes that can fail
  - Application decides whether to continue or exit

`apstat` Display Output Format

- **New feature: custom output from `apstat`**
- **Use the `apstat -f` option**
  - Name the columns as arguments to `-f`
  - In order you want
    - Comma separated
    - Quote to be safe
    - Same names as seen on output
    - Case sensitive
  - Only on single-table output
    - `apstat -a` (applications)
    - `apstat -r` (reservations)
    - `apstat -p` (pending applications)
    - `apstat -P` (protection domains)
    - `apstat -n` (nodes)
    - `apstat -G` (GPUs)
- The defaults for the system are set in `/etc/alps.conf` in the `apstat` section

apstat Display Output Format Example

```
users/rns> apstat -n |head
NID Arch State HW Rv Pl   PgSz     Avl    Conf   Placed PEs    Apids
  2   XT UP   B 24  1  1    4K 8388608  262144   262144   1 11461796
  3   XT UP   B 24  1  1    4K 8388608  262144   262144   1 11461796
  6   XT UP   B 24  -  -    4K 8388608       0        0   0
  7   XT UP   B 24  -  -    4K 8388608       0        0   0
 10   XT UP   B 24  -  -    4K 8388608       0        0   0
 11   XT UP   B 24  -  -    4K 8388608       0        0   0
 12   XT UP   B 24  -  -    4K 8388608       0        0   0
 13   XT UP   B 24  -  -    4K 8388608       0        0   0
 14   XT UP   B 24  -  -    4K 8388608       0        0   0
```

```
users/rns> apstat -n -f NID,State,HW,AHW,C/CU | head
NID State HW AHW C/CU
  2 UP   B 24   -    1
  3 UP   B 24   -    1
  6 UP   B 24   -    1
  7 UP   B 24   -    1
 10 UP   B 24   -    1
 11 UP   B 24   -    1
 12 UP   B 24   -    1
 13 UP   B 24   -    1
 14 UP   B 24   -    1
```

Node Health Checker (NHC)

- **Node Health Checker (NHC) can be configured to run at node boot, application termination, and reservation termination**
  - The distinction between application and reservation termination is new
    - CLE 4.1.UP01 includes a new *ini* style configuration file
  - NHC at boot is configurable and uses a different configuration file
    - The configuration file is in the compute node image
  - The NHC configuration file is in the default view of shared root
    - The new configuration file allows tests to be part of sets
      - The sets are *application* and *reservation*

Node Health Checker (NHC)

- **ALPS calls the script `xtcleanup_after` script on termination of a application**
  - Uses the NHC configuration file to determine action `/etc/opt/cray/nodehealth/nodehealth.conf`
    - Actions are, run tests each time (“always”) a job terminates or on error; the default setting is “errors”
  - When required initiates `xtcheckhealth` on the login or mom node where the application was launched from
    - These nodes maintain information on the application fan-out
      - If nodes in the application fan-out are unreachable, NHC can work around the problem
    - On the compute nodes
      - `xtnhd` invokes `xtnhc` which runs the tests specified in the NHC configuration file, output is sent to the console

NHC

- The tests determine if the compute nodes are healthy enough to support running subsequent applications
  - Healthy nodes are removed from the list of nodes to check
  - Nodes that do not respond to the initial tests are marked suspect and retested until they pass or time expires (35 minutes)
  - Bad nodes are removed from the application resource pool
    - Bad nodes can be marked admindown
    - Bad nodes can be marked unavail and dumped and rebooted
  - NHC tests include:

`Alps`	Checks that the `apinit` daemon is present on the compute node
`Application`	Checks that the previous application has exited from the compute node
`Filesystem`	Checks the health of the specified file system(s)
`Memory`	Checks that the amount of free memory available on the compute node
`Site`	Runs a plug-in script; disabled by default

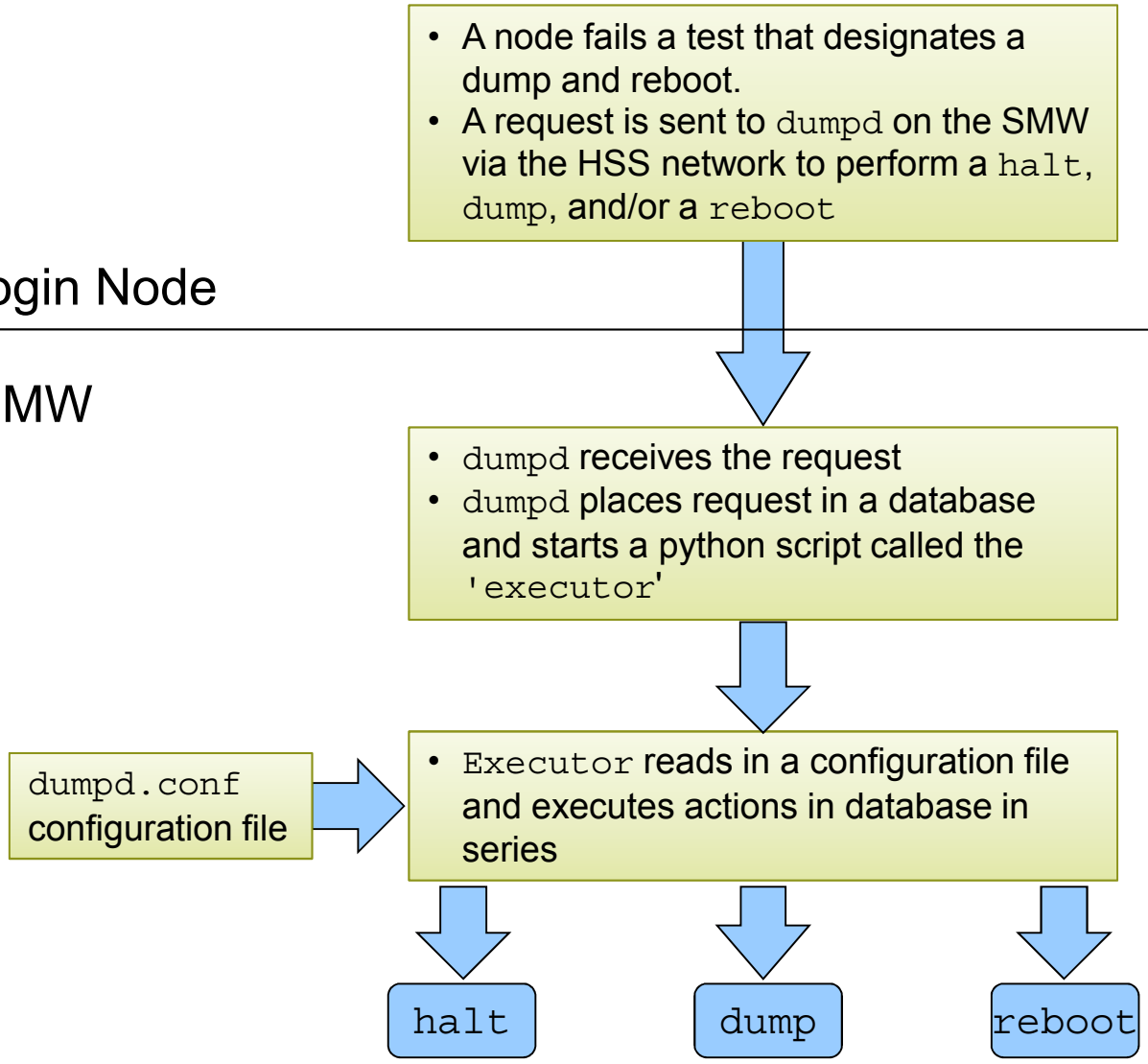
NHC

- **When NHC places a node into suspect mode it will now write an entry into a NHC table in the System database (SDB)**
  - Each entry contains information about the node, tests, and actions
  - A new daemon, nhcdbd (nhc-database-daemon) will run on the SDB node
  - nhcdbd will read the entries and coordinate actions requested by each NHC.
    - In order for an action to take place (reboot, dump, or dump&reboot), the following must be true:
      - All NHC instances have marked their db entries as ‘finished’
      - ALPS sees the node state as non-up
      - ALPS no longer has a reservation claim on the node

dumpd

- **dumpd is a daemon that lives on the SMW**
  - Part of the CLE software package (analogous to ldump)
  - Started by `xtbootsys` at system (or partition) boot
  - Listens for a specific `hss` event and runs commands based on the information it receives
- The configuration file is located on the SMW at: `/etc/opt/cray-xt-dumpd/dumpd.conf`
  - dumpd must be specifically enabled after a new CLE install by changing `enable` from `no` to `yes`
  - `dumpd` reads the `dumpd.conf` configuration file to find out the definitions of '`halt`', '`dump`', and '`reboot`'
  - Administrators can also add custom definitions

Overview of Dump and Reboot



dumpd Configuration File

```
# This variable functions as a quick on/off switch for all of dumpd
enable: no
[halt]
command: xtnmi --partition $partition $cname
max_cnames: unlimited
timeout: 60
[dump]
command: ldump -r xt-hsn@boot-$partition -o $dump_dir/$cname.$time
$cname
max_cnames: 1
simultaneous: 1
timeout: 1200
[reboot]
command: $dumpdbin/runpty xtbootsys --partition $partition --reboot
--compute-loadfile CNL0 --reason "Automatic reboot done by dumpd at
time $time" $cname
max_cnames: 50
simultaneous: 1
accumulation_time: 10
timeout: 1200
[myaction]
command: echo “myaction called” >> /tmp/output
```

Dumpd Administration Commands

- **dumpd-request**
  - A tool to allow administrators to request actions of dumpd
  - Command is present on both the SMW and system
    - On the system: `$ module load dumpd`
  - Example calls:

```
dumpd-request -a halt,dump,reboot -c c0-0c0s4n0
dumpd-request -a myaction,reboot -c c0-0s0s4n1
```

Note: The example assumes "myaction" is defined in dumpd.conf

- **dumpd-dbadmin**
  - SMW command that reads the database and displays the current actions, below is and example output:

```
dbid: 15582   cname: c5-2c0s2n0   partition: 0
queued:    current: dump  actions: halt,dump,reboot
APID 0 pid: 6102 requester: db-new   state: 0
orig time: 2011-02-13 09:38:32   last updater: db-new   \
       last update time: 2011-02-13 09:38:32
```

PE Installer

- Automate more of the installation process for PE products
- One configuration file that is used for whitebox, eslogin, esMS and internal login node installations
- Multiple Installation Scenarios
  - Install into multiple targets from the SMW
    - shared root on boot node
    - esLogin images on esMS server
    - unmanaged esLogins accessible to the SMW
  - Install on an esLogin directly
- Behavior driven by configuration file rather than command line arguments
- Configuration file checked for errors when installer starts
- Can install PE packages and run set_default scripts
- Detects pre-release versions of packages for Administrators to remove

PE Installer

- A YAML configuration file is used and would typically only need to be configure one time on your system
- YAML is an industry standard - http://www.yaml.org
- The installer includes template configuration files for:
  - CADE
  - CDT
  - The template files include comments describing each of the keywords in the file
- For more information on the configuration file see the *Cray Programming Environments Installation Guide* S-2372-102

Questions ?

Thank you for attending this presentation!