Case Studies in Deploying Cluster Compatibility Mode

Tara Fly, David Henseler, and John Navitsky
OS/IO
Cray, Inc.
Seattle, WA USA
e-mail: {tara,dah,johnn}@cray.com

Abstract— Cray's addition of Data Virtualization Service (DVS) and Dynamic Shared Libraries (DSL) to the Cray Linux Environment (CLE) software stack provides the foundations necessary for shared library support. The Cluster Compatibility Mode (CCM) feature introduced with CLE 3 completes the picture and allows Cray to provide "out-of-the-box" support for independent software vendor (ISV) applications built for Linux-x86 clusters. Cluster Compatibility Mode enables far greater workload flexibility, including install and execution of ISV applications and use of various third party MPI implementations, which necessitates a corresponding increase in the complexity of system administration and site integration. This paper explores CCM architecture and a number of case studies from early deployment of CCM into user environments, sharing best practices learned, with hopes that sites can leverage these experiences for future CCM planning and deployment.

CCM, CLE, DSL, DVS, IAA, ISV, MPI, ssh, rsh, cluster

I. INTRODUCTION

Cluster Compatibility Mode (CCM) extends the capabilities of traditional Cray Linux Environment (CLE) to allow many ISV applications to run without modification and without adding additional overhead to native Extreme Scalability Mode (ESM) applications. The traditional Cray architecture differs from traditional Linux clusters as CLE, by design, does not provide a full-featured Linux environment. In addition, on larger scale systems, interactions with ESM and configuration scalability should be considered when configuring the system to support CCM. This paper provides a design overview of CCM, an overview of the architecture of the Cray system components that interact with CCM, an overview of the CCM technical components, considerations for scalability, and some examples of site configurations.

II. DESIGN GOALS

The design goals of CCM were threefold: to provide an environment to run independent service vendor (ISV) applications out-of-the-box without modification, to ensure the capabilities necessary to run these ISV applications without sacrificing the performance of traditional Extreme Scalability Mode (ESM) applications, and to continue to optimize performance moving forward. ISV applications generally consist of a pre-compiled binary, packaged with a specific version of a third-party MPI, often requiring a license for execution. Dynamic Shared Libraries (DSL) provides the foundation for pre-compiled binaries. Then, with the addition of shared library support in Cray's Compute Node Linux (CNL), it is no longer necessary to compile specifically for Cray XE/XK systems. Cray provides for compute node IP-connectivity through Realm Specific IP Addressing (RSIP), which is used to address licensing needs of the ISV applications. CCM is then responsible for addressing additional ISV application needs: specifically in the areas of MPI-launch support, and Linux Standard Base compliance. Third-party ISV codes use a number of different launch mechanisms for job fanout: typically rsh, ssh, or Linda (over rsh or ssh). In addition, many applications expect writeable /tmp, dbus, and POSIX shared memory segments to be persistent through the lifetime of a job. Finally, CCM continues to optimize for performance. The goal is to approach native gemini performance below 2048 PEs, which bounds typical ISV application scalability. CCM was first introduced in CLE 3 with support for TCP. The ISV Application Acceleration (IAA) component was added to CLE 4.

CCM does not provide a runtime environment to users, and was never designed to provide a development environment. Where users have access to source code, Cray provides the Cray Programming Environment. Cray's programming environment is extensively optimized for scalability and performance, and uniquely tuned to the Cray systems architecture. Also, CCM does not attempt to provide the infrastructure to allow general service provisioning on compute nodes. Rather, CCM strives to provide the minimal excess service overhead to still fully support any ISV Application.

III. ARCHITECTURAL OVERVIEW

A. CLE Dynamic Shared Libraries

To understand how to deploy CCM on a Cray, it is important to understand the Cray system architecture. Cray uses a single shared-root file system, projected over Network File System (NFS) from the XE/XK boot node. The shared-root file system is mounted read-only on all XE/XK service nodes. If DSL has been installed, the shared-root is projected by Data Virtualization Service (DVS) to the CLE compute nodes. The Compute Node Root Runtime Environment (CNRTE) projects the shared root to compute node images. CLE users have the option of launching their jobs in native

CLE low OS-noise environment, or using DSL to access the shared-root through a CRAY_ROOTFS environment variable. The DSL and CNRTE components are optional on the XE/XK systems, though required for CCM. CCM runs on top of the DSL root, although it makes no requirements on the system default.

Fig. 1 shows a DSL typical implementation with the shared-root projected to the compute nodes. DVS clients running on each CLE compute node can be configured with load-balancing. In a DVS load-balance environment, the CLE DVS client will select the primary server based on a node network identifier (nid). If a DVS server fails, the clients fail over to another DVS server which provides resiliency and scalability. DVS clients support both page caching and attribute caching for additional performance optimization. DVS servers can be run on XIO nodes, or repurposed compute nodes. Repurposed compute nodes allow sites to use XE/XK compute nodes as PCI-less service nodes; these nodes must be statically booted as service nodes, and are removed from the pool of available compute node resources. DVS is best configured by the CLE installer.

B. *Shared Root*

All nodes in the Cray system share a single shared-root file system. Traditionally, Linux services are managed with configuration files in /etc, and enabled using the insserv and chkconfig commands. To provide the ability to run different services and have different configurations on different nodes, the XE/XK system establishes an inheritance hierarchy for the /etc file system on the shared-root. Fig. 2 presents a simple view of shared-root configuration.

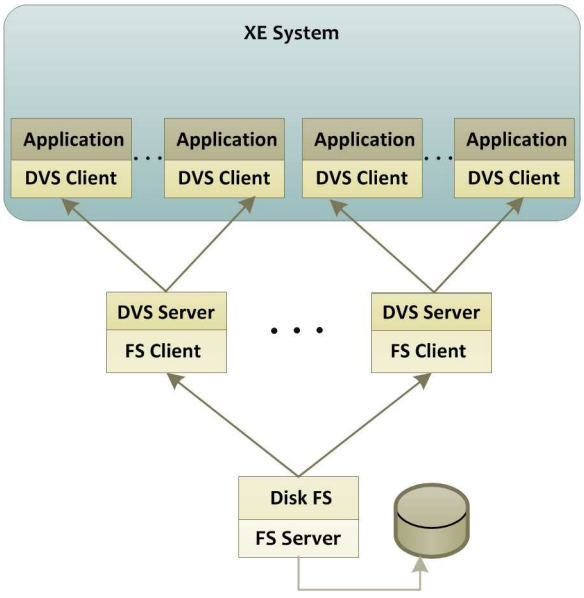

Figure 1. Shared-root projection to compute nodes over DVS

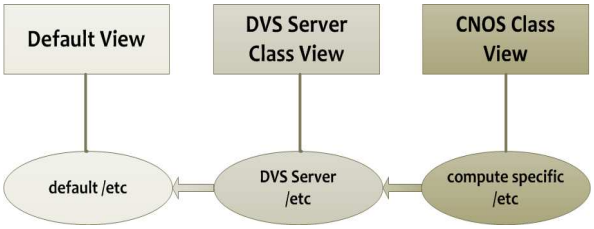

Figure 2. Shared-root specialization

At the root of the hierarchy is a common default view, which is superseded by a class view and a node specific view for each node on the system, if available. A given system node can belong to exactly one system class. System classes allow the administrator to make configuration file changes common to multiple nodes providing the same system functions. For example, an administrator might want to set up name services in the same fashion on all nodes functioning as login nodes through a login class view, whereas external Ethernet IP configuration files could be managed in the node specific views for a given XIO node. All configuration files exist in a global system default view of /etc, which is a directory containing a set of symbolic links to the original distribution contents of /etc. The CLE installer sets up the symbolic links in the default view at initial install time. Afterwards, contents of these links must be managed through the xtopview utility. All Cray developed software components deliver their configuration files to /etc/opt/cray on the shared root, thus providing system configuration through a single xtopview interface.

C. *Compute Node Root Filesystem*

DVS servers work by projecting the entire file system mounted on the DVS server to the CLE compute environment, which is in turn, mounted to a specified path location (/dsl by default) on the compute nodes. Users launch ESM jobs to compute nodes with the aprun command and aprun performs a chroot into the DSL root, and then launches the specified executable. Since administrators may want a different system administration configuration for compute nodes than they desire on the DVS server, Cray implements a special class, the cnos class, to provide management of /etc files mounted on the compute nodes. The cnos class is common to all DSL jobs, including CCM jobs – this is an important factor in some of the administration case studies provided later in this paper.

D. *RSIP*

RSIP, although not strictly required for CCM, enables CLE compute nodes to access external network resources. An RSIP client running on the CLE compute node interfaces to an RSIP server through an IPIP tunnel. With RSIP, application packets are forwarded over the IPIP tunnels while protocol traffic is distributed over the HSN. Administrators configure a port range that can be used with RSIP. The

number of ports available to any given RSIP client is equal to the lesser of 255 or:

num_rsip_servers × num_ports_in_range / num_clients (1)

The total client ports limit the number of outbound connections from any given compute node.

IV. Cluster Compatibility Mode Architecture

A. Session Initialization

When a batch job is submitted, a batch server contacts its scheduler, which issues an Batch Application Scheduler Interface Layer (BASIL) RESERVE request to the application level placement scheduler (ALPS) to make a claim on the requested node resources so that they are available to the job when it executes. The exact contents of the packet vary by BASIL protocol version, but always include a ALPS reservation identifier, and a cookie. The batch server then sends an apbasil CONFIRM request to associate the claim with a specific process group (PAGG) on the login node, and to bind the resources to the batch system for the duration of the reservation. All CCM-compatible workload managers provide support for a root-owned pre-execution hook that is called after the resources are bound to a specific batch reservation. If a job is to execute in Cluster Compatibility Mode, the pre-execution hook, or prologue performs several steps during initialization, including:

- Validating that a specific batch job submission is requesting CCM. As of 4.1UP03, CCM provides support for configuring CCM for all jobs submitted to specific batch queues, or alternately by using custom resource configuration if supported by the WLM vendor
- Creating a nodelist containing, on separate lines, one host entry for each PE in the job reservation
- Providing correct and transparent nodefile entries on the login node and all compute nodes in the job
- Generating ssh key files for the user, if these do not exist
- Populating .rhosts in the user $HOME directory when CCM_ENABLERSH has been set for the system
- Transferring the files necessary to support and isolate the compute node CCM service configuration to a specific user/job instance
- Fanning out to all compute nodes in the job using the xtxqtcmd binary provided with the nodehealth rpms to prepare the environment

B. Environment Setup

Environment setup performs the following actions:

- preparation of all of the configuration files required by sshd and xinetd, staging these to a subdirectory of the compute node's /var filesystem
- creation of a temporary environment using bind mounts to provide a writeable /tmp, writeable /var/tmp, /dev/random, a CCM-specific /var/run directory, pseudo-terminal support, MPI DAPL configuration, dbus protocol support, POSIX shared memory support, sshd configuration and xinetd configuration.
- providing the ability to extend CCM bind mounts to accommodate site specific configuration with a /etc/opt/cray/ccm/ccm_mounts.local configuration file

The described configuration persists for the lifetime of the batch reservation. At the end of the batch job execution, the workload manager calls a post-execution hook, or epilogue which is responsible for tearing down the CCM configuration, by:

- validating that the specific batch reservation is targeted at CCM
- creating a lock on the reservation to prevent multiple epilogue execution in the case that the script is resubmitted
- fanning out to each compute node in the job using xtxqtcmd and attempting to umount any CCM mounted file systems
- stopping all services started by CCM
- removing all temporary files staged by CCM, if the virtualized environment was successfully removed
- checking on the health of the node, marking it administratively down if the CCM environment was not properly removed

C. Node Health Checker Interaction

As of CLE 4.1UP03, CCM uses the node health checker to monitor for application exit, and after successful exit of all applications on the compute node, removes temporary data and bind mounts. The post-execution hook calls the node health checker (NHC) directly and, after checking for a small amount of time, it marks the specific node SUSPECT, exits the post-execution script, and allows batch system to release the claim on the nodes. NHC then can asynchronously monitor for node recovery and restore the nodes to service once CCM has been torn down. These extensions were made to provide greater resiliency to out of memory situations, file system cache flushing and application core dumps and to provide greater portability between WLM versions and vendors.

CCM continues to evolve and adapt to the needs of ISV applications, and site implementation specifics, with the goal of supporting any ISV application and providing an increasingly turn-key environment for standard deployments.

V. Job Execution

A. Linux Cluster JobLaunch

On a traditional Linux cluster, a user requests an allocation from a workload manager. The workload manager

allocates node resources for the job, and then launches the job on one of the nodes. The MPI programming model provides a message passing interface to provide inter-process communication. Consider the simple case of an MPI hello world, submitted on two nodes, launched with:

mpirun –machinefile $NODEFILE –np 2 mpi_hello

The mpirun command reads the file specified by *--machinefile*, processes a list of nodes, distributes the executable to *-np* nodes using ssh or rsh to provide this transport and executes program *mpi_hello* on each of these nodes. In traditional clusters, the batch system is responsible for job allocation and application placement.

B. Cluster Compatibility Mode Job Launch

On a XE/XK system, the batch system is responsible for resource scheduling, but ALPS provides application placement. Unlike the cluster model, application launch occurs on a login node that is not part of the compute cluster. The Cray CCM architecture implements a similar model to cluster job launch, adapting to ALPS placement model. This architecture is shown in Fig. 3.

CCM job launch is initiated from an XE/XK login node. The application user submits a job through a workload manager, which then requests the resources associated with the job, allocates the nodes, and executes the script on a MOM / sbatchd node. A job script includes any job and environment setup, and a ccmrun statement. The ccmrun command is responsible for initiating cluster services and for placing a single copy of the application launch command onto the first node in the job claim.

To set up a cluster environment, ccmrun launches a single copy of special program called ccmlaunch passing as argument the application binary or command, and all associated launch arguments.

The ccmrun command bypasses binary transfer of the target executable, instead passing the application path to an apinit process on the compute nodes.

Apinit sets up the job context and gemini context (including a PTAG and a cookie), which allows the placed job to access the HSN. Apshepherd then executes a chroot into the DSL root, and executes ccmlaunch. The ccmlaunch command starts a selected set of system services on the compute nodes. These services include nscd, rpcbind, an alternate sshd listening on port 203, and optionally xinetd to provide rsh, rlogin, and rexec services.

All services are started specifically by running the /etc/init.d service scripts in the compute node CNRTE root. A full Linux init sequence is not performed on the compute node however; rather CCM anticipates any requisite services for the limited set of services it provides. System administrators may also enable NIS for CCM compute node jobs. NIS must be configured globally in the ccm.conf configuration file as ypservices will fail and delay startup if not properly configured.

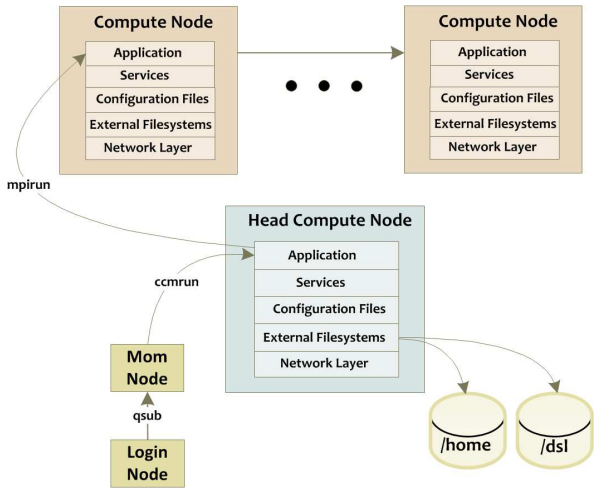

Figure 3. CCM Job Execution

Once service initialization is complete, ccmlaunch sets up a sync barrier to ensure that daemons have completed initialization on all compute node jobs, and then queries ALPS for the local Programming Environment (PE) rank. If ccmlaunch determines itself to be executing on PE[0], it acts in the role of CCM head node. The application launch command provided to ccmrun is then placed on the node for execution, and ccmlaunch monitors for job termination. On all other PE ranks, ccmlaunch sleeps until the application job has terminated. Without this sleep, aprun would detect the exit of the application on the ranks, and initiate a teardown of all job nodes.

Figure 4 shows an MPI application launch in the CCM environment; the dashed line represents the synchronization barrier used for service initiation. In this illustration, the application processes are direct children of the CCM-provided sshd. All processes are confined to the ALPS-managed PAGG, allowing them access to the gemini network. As the gemini context is unique to each job launch, all processes in the PAGG are torn down after every ccmrun command.

VI. Remote Application Launch

CCM provides password-less ssh and rsh setup to provide turn-key support for ISV application launch, and to prevent accidental interference with other user jobs. This section describes how CCM configures the environment for remote application launch.

A. Secure Shell

Standard MPI application launch requires password-less key exchange between the nodes. Traditional Beowulf-type clusters require the user to set up a password-less key pair on the system-level node. On a Linux-based distribution, this is done by running ssh-keygen.

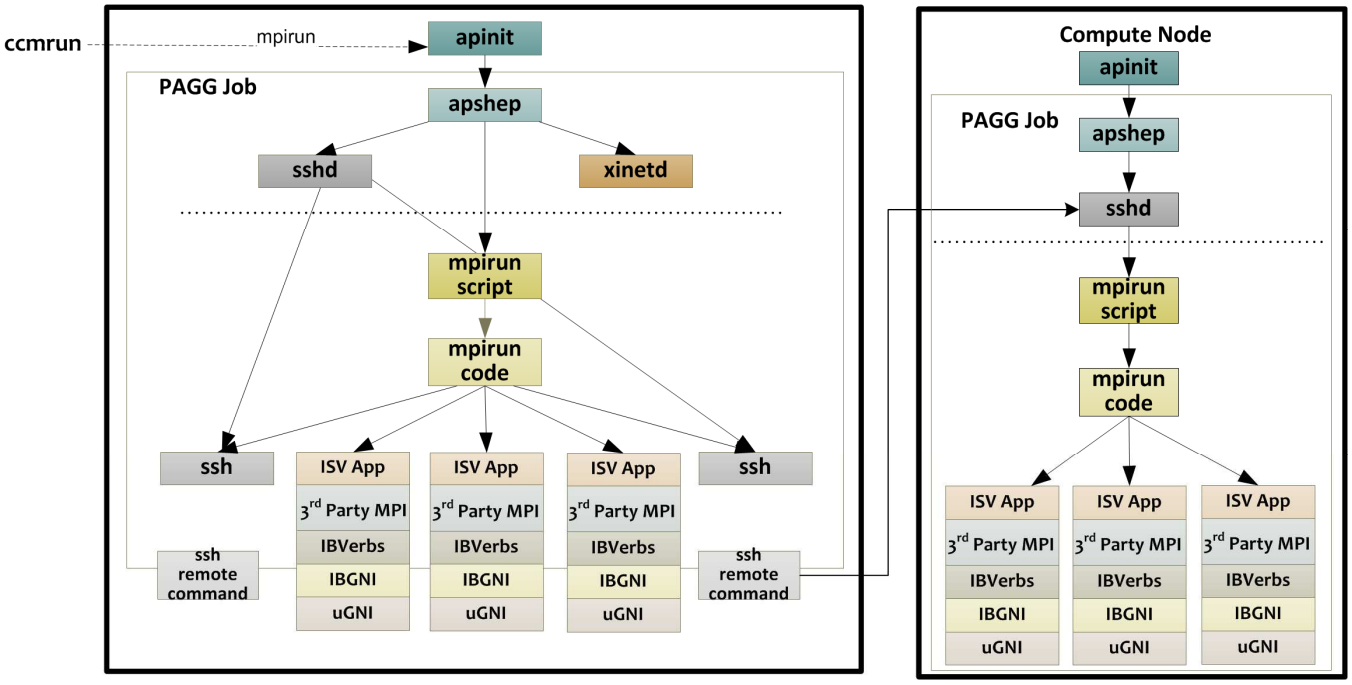

Figure 4. Simplified CCM Job Launch

The resulting keys are stored in the $HOME/.ssh directory; additionally, $HOME/.ssh must have 0600 permission. CCM attempts to create password-less ssh key pairs on the login node, if such keypairs do not exist. Otherwise, the user is responsible for generating ssh key pairs. On the compute side, the sshd is configured to allow key-based authentication. In the CCM implementation, the CCM startup process transfers the public key of the user to the job compute nodes. These compute nodes place the public key file in a path file location. The CCM sshd is configured to accept the keypair through the sshd_config AuthorizedKeyFile in lieu of the authorized_keys in $HOME/.ssh. This allows CCM to isolate itself from the user's login environment. CCM derives its sshd_config from the /etc/ssh/sshd_config file in the cnos class view, and appends only a small set of options. This allows the CCM ssh to more easily co-exist with site-built ssh alternatives. The sshd service is started with /etc/init.d/sshd, so it can support site-built packages.

B. Remote Shell

The rsh protocol is provided by xinetd. CCM enables rsh, rexec and rlogin and then starts rpcbind and xinetd services. CCM populates a local CCM file with correctly formatted .rhosts contents, and then mounts this file on top of $HOME/.rhosts. For sites wanting to use rsh, the administrator must configure CCM_ENABLERSH in the ccm.conf file, and the site must allow a user to create a $HOME/.rhosts file with mode 0600 permissions, as well as allowing root execute on $HOME. As the rsh protocol does not encrypt its communications and has weaker password-less authentication mechanisms, site administrators can choose to disable CCM rsh support entirely.

VII. CLUSTER COMPATIBILITY MODE JOB LAUNCH

To support ssh and rsh, a user must be able to execute a login shell on the compute nodes. Compute Node Linux does not require either user accounts or home directories. Supporting users requires new configuration and planning for the system administrator. This section provides some examples of working configurations at sites.

The most common implementations of providing /home to CCM are by sourcing /home on lustre and or hosting /home off of an external NFS file server. The home directories on any XE/XK internal login / login gateway nodes are typically the same as the /home directories mounted on the compute nodes and any automated user environment configuration makes this assumption. For external login nodes, the batch submission routes the request to an internal batch node manager, and all CCM configuration is initiated from within the XE/XK system. In theory, home directories for internal and external login nodes should not need to be the same from a CCM perspective. Separating /home on internal and external nodes may be an impractical configuration for other reasons and subsequent deployment use cases assume a transparent /home.

An ssh connection will authenticate a user using Name Service Switch (NSS), specifying that CCM use static

password, NIS, or compat semantics. Node specialization will first look for /etc/nsswitch.conf in the cnos class view, then the node specialized view of the DVS server, the class view of the DVS server and finally the default view. Sites using a login class view will need to separately administer any password management files. As a tradeoff, sites gain greater configurability to manage CCM user authentication to meet scalability and security requirements.

Fig. 5 depicts a site with external login nodes, and using LDAP for user authentication. Transparent /home is provided from an external NFS volume throughout the system. A common, external LDAP server provides name service resolution.

Here, the DVS server mounts an external NFS /home directory, and then projects that directory to the compute node. The same directory is also mounted on the XE/XK Login node and the esLogin node. The compute node here is configured to talk to an RSIP server, which, in turn, can speak to an external LDAP server. The login gateway and esLogin can talk to the LDAP server directly. RSIP on the compute nodes is required in this configuration to authenticate the user.

Fig. 6 illustrates a compute node talking to an NIS slave server running on an I/O gateway node. The gateway node has the necessary IP connectivity to talk to the NIS master. This configuration eliminates the need for an RSIP server to support name services on the compute node. Instead, it distributes the load to the NIS server. On an XE/XK system, the NIS server must be located on the high-speed network (HSN), due to a RSIP and sunrpc compatibility limitation. The same configuration can be used to support LDAP to limit LDAP server load.

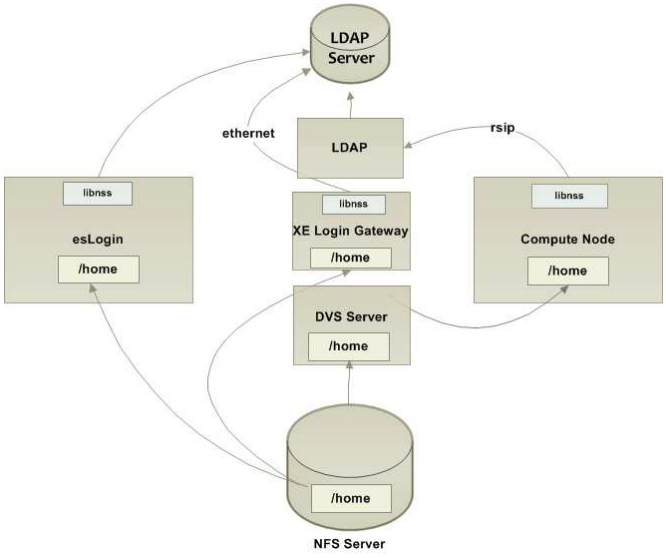

Figure 5. Transparent home directories with external LDAP server

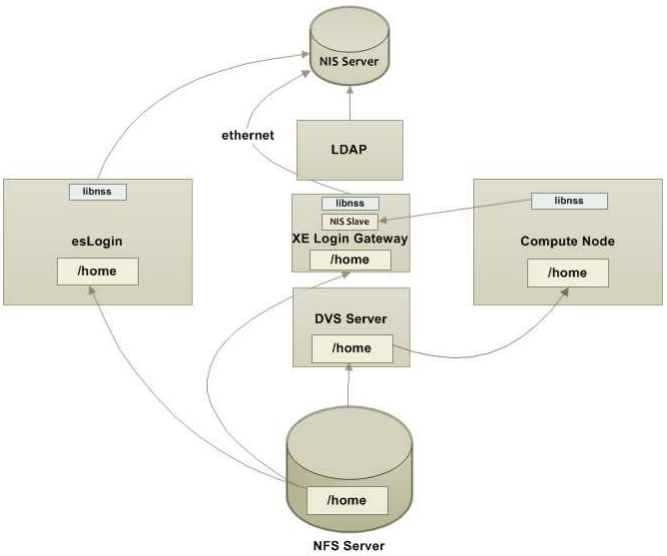

Figure 6. Compute nodes talking to internal NIS slave

CCM attempts to limit name service activity by starting up a name service caching daemon for each job instance. Since nscd starts on every job launch, each new job launch may make a single request to the name server. Since job launch is simultaneous, initial job launch can potentially put a heavy simultaneous load on the name servers. Site policy should consider this load and their system size when choosing the name service authentication method and determining the number and placement of name servers. Some configurations are only appropriate for smaller systems. Static password authentication will provide the maximal scaling and performance.

A final consideration when using LDAP or NIS with CCM is that the configuration files are shared between CLE ESM jobs running on the DSL root and CCM jobs, as these files will share the same /etc/nsswitch.conf. As this poses distributed denial of service potential in a large site, the following alternative was devised. As mentioned previously, an administrator can add additional CCM mount points. These mount points are specified in a configuration file: /etc/opt/cray/ccm/ccm_mounts.local. An administrator can specify a separate configuration file to be used only for CCM jobs, without modifying the configuration seen when running ESM jobs with a simple entry like:

```
/etc/nsswitch.conf.ccm  /etc/nsswitch.conf  bind  0
```

VIII. CSA Accounting Interactions

Experience has shown that improper CSA accounting can break CCM jobs using MPI with the InfiniBand Verbs API. When a job is initiated on a Cray XE/XK system, the PAGG job ID is first obtained by the batch system that launches the job. For interactive jobs, the PAGG job ID is obtained at login time by the pam_job module. ALPS uses

the PAGG job ID to uniquely identify the job for accounting purposes. CSA provides an ioctl interface to pass job accounting data to the CSA kernel software, which then stores the data in pacct. The sshd configuration on an XE/XK login node is configured to use pluggable authentication modules (PAM), and to include pam_job during session management, triggering the creation of a new PAGG job ID on every login. The PAGG is a nearly inescapable job container that contains all user processes. The only time a new PAGG would be assigned is when a new login session is created by sshd, which is one of the mechanisms that third-party mpirun and mpiexec uses to distribute work to other nodes in a cluster. If mpirun is configured to use ssh and the pam_job is loaded, the new ssh session and all children of that ssh session will be placed in a new process aggregate group.

As previously described, during application launch, ALPS associates the PAGG job ID with a unique gemini PTag, which is used to establish the gemini protection domain. Only jobs within the specific PAGG can access the high-speed network. If an application escapes the job container, it will get errors when attempting to use the InfiniBand Verbs API.

IX. The NFS Root Squashed Environment

In many sites, administrators choose to serve home directories with NFS root_squash enabled. In an NFS root squashed environment, the NFS server assigns the anonymous user id *nobody* to all root user access initiated by an NFS client. DVS preserves all attributes when projecting an NFS volume to clients running on a compute node, so client accesses from compute nodes respect root squashed home directories. The initial CCM implementation created temporary user key files for each CCM session, mounted on top of the existing keys in the user $HOME/.ssh directory. Since password-less ssh requires 600 permissions on .ssh, the contents of this directory cannot be viewed on the CLE compute node, and this initial implementation was discarded. Today, the user public key from the login node is transferred to the compute node, and an alternate authorized_keys file is specified to avoid this limitation. Longer term, prototyping is being done on shifting to a host-based authentication in the CCM context only, with a return to dynamic key generation.

A final implication of root squashed home directories is the impact on sites using rsh for MPI job launch. Due to the previously discussed interaction of pam_rhosts and the shared-root, CCM relies on a correct .rhosts file in the user home directory. For CCM to provide temporary and transparent rsh services, root must be able to execute within the user home directory. If home directory permissions are correct, CCM will bind a correct rhost file on top of .rhosts in the home directory. The CCM rhosts file also restricts password-less rsh to the nodes assigned to the job, preventing accidental interference with other jobs on the system. If security policy does not allow these permissions, users will need to stage their own .rhosts file with the correct contents or use ssh launch as the alternative.

X. Graphical User Interface Support

The CLE Linux distribution includes the X Window System (X11) for GUI support. As the goal of CCM is to provide the capability to run any pre-compiled binary, it also provides a mechanism for graphical user interface support through ssh forwarding and ccmlogin. To get end-to-end X11 tunneling back to a desktop, a user connects to the XE/XK login node in an ssh session with X-forwarding enabled. Next, the user requests an interactive session from the batch system, indicating to the WLM that it should propagate environment to a CCM job. Finally, the user connects to the head node of the CCM job, requesting ccmlogin to propagate the environment to the compute node. Like ccmrun, ccmlogin will start up services for the session, and then will initiate an ssh session to the numerically first node of the job reservation. The user logs into the node, completing the X-tunnel. Fig. 7 illustrates the X-forwarding process.

Providing interactive user shell on the compute node does pose challenges to a cluster administrator. The batch system needs to be configured to place the batch interactive shell on the same node, or must be capable of propagating the DISPLAY environment variable to the new shell. Users can inadvertently forget to exit the shell, and leave system resources consumed. Administrators may want to consider enabling ssh keepalive, or advising interactive users to do so, to prevent sshd from closing the connection. Batch wall clock timers can be used to terminate interactive users after a specified duration of time.

XI. Workload Manager Integration

The CCM package includes batch prologue and epilogue scripts that need to be called by the WLM. These script callouts can be integrated with existing batch hooks. Both CLE and WLM documentation provide information about configuring scripts with the batch system. To enable on an XE/XK system, the administrator must configure the batch system to call these scripts and must designate to CCM specific target batch queues or a specific set of batch resources as allocated to CCM jobs. In addition, the resources or queues must be configured in the WLM.

Each supported WLM has the capability of restricting the number of nodes to a given queue, or the number of total batch resources of a particular type that can be requested.

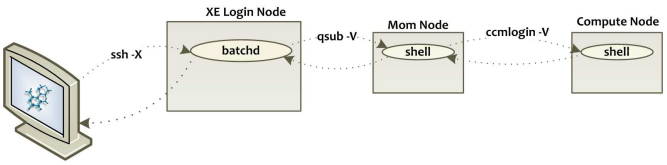

Figure 7. X11 Forwarding through batch system

Administrators can use these options to apply any number of restrictions when deploying CCM on site: e.g. the maximum size of a CCM job, the number of simultaneous CCM jobs allowed at a given time, the time of day jobs can be submitted to CCM, restricting CCM to a certain subset of the system, the total number of nodes that can run CCM at any given time. Administrators can set aside a smaller set of resources for interactive CCM users, and put aggressive timeouts to prevent users from accidentally tying up machine resources.

CCM provides a few additions to help support end user application launch scripts. One such feature is transparent node lists through the cluster. Without CCM, the workload manager nodes file on the login node is incorrect, containing the hostname of the mom node rather than the lists of hosts in the job. During initialization, CCM corrects the nodes file to reflect all of the PEs in the reservation, as would be seen on a standard Linux cluster. Additionally, CCM makes sure that the correct nodes file is present on the compute nodes.

Cray XE/XK systems do not provide batch node managers on the compute nodes. Where industry workload managers scale to tens of thousands of nodes, the ALPS architecture is designed to scale to hundreds of thousands of nodes. There are a few customer visible side effects; customer scripts can run qstat commands on the master mom node, but not individual job nodes, and MPI applications that were compiled to support Moab/Torque native launchers need to explicitly disable native launch through MPI command options: "--mca plm ^tm --mca ras ^tm –x".

XII. Conclusion

Cluster Compatibility Mode allows customers to run their existing applications on an XE/XK system, while not compromising traditional extreme scalability mode workloads. Cray continues to work actively with ISVs to provide Cray native ports where appropriate, and encourages Customers should use Cray Native applications when available, as these always provide the optimal performance and scalability. ESM provides optimal performance and scalability when customers have access to application source code. CCM addresses the need to run applications that have been built for standard Linux x86 clusters. Looking to the future, CCM will continue to work to optimize performance, and adapt to customer needs.