Supporting Multiple Workloads, Batch Systems, and Computing Environments on a Single Linux Cluster

Larry Pezzaglia
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA USA
lmpezzaglia@lbl.gov

***Abstract*—A new Intel-based, InfiniBand attached computing system from Cray Cluster Solutions (formerly Appro), at NERSC, provides computational resources to transparently expand several existing NERSC production systems serving three different constituencies: a mixed serial/parallel mid-range workload, a serial, high-throughput, High-Energy Physics/Nuclear Physics workload and a mixed serial/parallel Genomics workload. This is accomplished by leveraging a disciplined image building system and CHOS, a software package written at NERSC, comprised of a Linux kernel module, a PAM module, and batch system integration, to concurrently support multiple Linux compute environments on a single Linux system. Using CHOS, this new computing system can run workloads built for the hosted clusters, while simultaneously maintaining a single consistent, lean, and maintainable base OS across the entire platform.**

The origins, design, and implementation of the CHOS utility, and how NERSC's use of this utility in production has resulted in a substantial elimination of system management overhead and user impact, will be discussed in detail.

***Keywords*-image management, provisioning, cluster management**

I. Introduction

The National Energy Research Scientific Computing Center (NERSC), located at Lawrence Berkeley National Laboratory (LBNL), is the production scientific computing facility for the US Department of Energy (DOE) Office of Science. NERSC provides computational resources to support the scientific activities of over 5000 users. In addition to Hopper, a Cray XE6, and Edison, a Cray XC30, NERSC also supports three x86_64-based mid-range computational systems[1]:

- **Carver**: An IBM iDataPlex cluster of over 1000 nodes running Scientific Linux (SL) 5.5, interconnected with QDR IB, and supporting a general-purpose serial and parallel workload. The batch system is TORQUE+Moab, and the nodes are managed with xCAT.
- **Genepool**: A commodity Linux cluster of over 400 nodes running Debian 6, interconnected with 1GbE and 10GbE, supporting serial and parallel workloads from the DOE Joint Genome Institute (JGI). The batch system is Univa Grid Engine (UGE), and the nodes are managed with Perceus.
- **PDSF**: PDSF (Parallel Distributed Systems Facility) is a commodity Linux cluster, with over 200 compute nodes as well as over 1 PB of internal storage. PDSF is interconnected with 1GbE and 10GbE and provides SL 5.3 and SL 6.2 computational environments to support exclusively serial, high-throughput workloads from High Energy Physics and Nuclear Physics experiments. The batch system is UGE, and nodes are managed with xCAT. PDSF has been in continuous operation since 1996.

Each of the aforementioned midrange systems was designed and deployed to support the needs of a distinct NERSC user constituency. To accommodate demand for increased computational capacity by all three constituencies, NERSC elected to deploy a single new hardware platform, known internally as Mendel. Mendel would need to be flexible enough to meet the needs of all three user constituencies, and expandable enough to meet future needs. Additionally, a unified configuration and platform management software environment across the entirety of the new system was strongly desired to improve consistency and maintainability.

Mendel expands these three "parent systems" in a fashion nearly transparent to the users of those systems. This is accomplished by assigning groups of Mendel nodes to each parent system, and then modifying the software and firmware configuration on these nodes to match parent system requirements.

In addition to the aforementioned computational systems, NERSC provides a number of support services, including NX[2] and MongoDB servers. The Mendel platform provides housing for these services as well.

II. Requirements

Ensuring a seamless expansion of the parent systems imposes several requirements on the Mendel platform:

[1]A complete list of computational systems at NERSC is available on the NERSC Web site at http://www.nersc.gov/computational-systems.

[2]NX is a software package to optimize performance of the X Window System over low-bandwidth and high-latency network links.

- All source code that compiles on the parent system must also compile on the expansion system.
- Executable code compiled on the parent system should function on the expansion system. Users should not be burdened with a recompilation requirement. However, recompilation could be recommended to take advantage of new hardware features of the Mendel platform. For example, OpenMPI applications from the Genepool system built to use the OpenMPI TCP transport should continue to function using the IPoIB interfaces on Mendel nodes, but a recommendation could be made to rebuild the application against an OpenMPI supporting the IB transport.
- All job submission scripts used on the parent system must also work on the expansion system.
- Users should not be required to modify their workflows, unless they desire to do so to take advantage of new features of the new hardware.

III. Approach

One approach to the challenge of expanding these parent systems onto the Mendel platform is to boot each subset of Mendel nodes into OS images that mimic the parent system. This requires duplicating the boot images from each parent system and adding support for the Mendel hardware, platform management tools, and configuration management schemes. Each image would be maintained separately over the lifetime of the parent system. Advantages of this approach include a simple node boot process and maximum similarity in software configuration between the Mendel expansion and the owning parent system. However, this approach introduces problematic maintainability issues:

- Deploying a software change to the entire Mendel platform would require individual changes to each boot image. System administrators would need to analyze the impact of every change on every base OS configuration, multiplying the maintenance burden.
- Versions of core system software, such as the Linux kernel, Mellanox OFED, and GPFS, must be kept synchronized across the entirety of the Mendel platform. To avoid version desynchronization, system administrators would be required to update every image whenever an upgrade of these components was merited, further increasing the maintenance burden.
- As parent systems run different Linux distributions, the subsets of Mendel would be substantially dissimilar. Each subset could encounter issues unique to its OS configuration. Furthermore, each distribution has unique configuration file layouts, administrative recommendations, and upgrade guidelines. Support for each distinct base OS configuration would need to be implemented in all system administration scripts, practices, and operational procedures.

Exacerbating these issues is the need for more than three images to to support the three parent systems. Distinct node roles in parent systems, such as compute node and login node roles, can require different images. Additionally, NX, MongoDB, and support nodes each present distinct software needs and would require separate images. All images would need to be manually kept compliant with site and system policies. As the number of images increases, the maintenance burden grows dramatically. Consequently, this approach scales poorly as the number of distinct node configurations increases, and it was judged to be a suboptimal management method for the Mendel platform.

Instead, NERSC architected a layered model providing a framework for accommodating all the computational and support service workloads on top of a unified hardware and software platform. NERSC developed a single base OS image to run on every Mendel node, and designed a structured "node differentiation" process to dynamically align nodes to parent system configuration needs at boot time. Differentiated nodes belonging to compute systems run batch execution daemons which accept jobs scheduled by the batch system software located on the parent system. The workload environments are then provided on top of the differentiated node environment.

Figure 1 shows the layered Mendel combined cluster model. The Base OS layer is responsible for core system software, such as GPFS, Mellanox OFED, and SSH. The Boot-time Node Differentiation layer holds responsibility for keeping each node's system configuration, such as authentication and authorization configuration and mounts of appropriate shared filesystems, compliant with parent system requirements. Specialized environments for user applications are presented at the CHOS layer.

This approach provides separation between the base OS and the user-visible job environment. Consequently, the construction of the base OS can be optimized for size, maintainability, and system administrator convenience, while the design of the computational environments can focus exclusively on the desires of the users.

The layered approach also increases system complexity. A thorough understanding of the Mendel boot-time node differentiation process, the distinction between the base OS environments and the user computational environments, and operational knowledge of all software tools involved are all essential knowledge for system maintenance. Consequently, the site-specific knowledge requirements for administrators and user support staff are increased. While this imposes some additional burden on site staff, the advantages of this approach were found to significantly outweigh the costs.

IV. Implementation

A. Hardware

The Mendel hardware and networking platform serves as the lowest layer of the layered Mendel combined cluster

model. The hardware platform is provided by Cray Cluster Solutions and designed using a "Scalable Unit" (SU) model, which provides a framework and concrete architectural details for disciplined expansion over time. Further expansion can be accomplished by purchasing and integrating additional SUs. As of April 2013, the Mendel system contains over 400 compute nodes.

The production network interconnect is FDR InfiniBand, provided by Mellanox SX6518 and SX6036 switches. Each production node has a 4X FDR link to the IB fabric. Additionally, a set of 1GbE Ethernet switches provides a separate network for node booting and management.

The compute nodes used on the Mendel platform are half-width Intel servers based on the Intel Jefferson Pass and Washington Pass platforms[3]. These half-width nodes are mounted in a set of H2000JF 2U chassis, each of which houses four compute nodes, two redundant 1600W power supplies, and three 3.5" SAS hard disk bays for each node. Each compute node has two 8-core Intel Xeon E5-2670 processors mounted on an Intel S2600JF or S2600WP system board, which includes on-board FDR IB.

Management, login, and service nodes are provided by 2U Intel servers with two 8-core Intel Xeon E5-2670 processors mounted on S2600GZ system boards. Large memory compute nodes with 512GB or 1024GB of RAM are provided with servers based on the Intel S4600LH2 platform with four 8-core Xeon E5-4650L processors. The management, login, service, and large memory nodes all include Mellanox ConnectX-3 VPI IB HCAs.

B. Software

A number of software tools were selected, created, or enhanced to implement this layered model. Some of these utilities had been developed by NERSC staff to fulfill previous needs.

- **CHOS**[4] ("CHroot OS"): A utility developed at NERSC to concurrently support multiple Linux environments on a single Linux system. CHOS is distributed under a modified BSD license.
- **xCAT**[5] (eXtreme Cloud Administration Toolkit): A widely used platform management tool capable of managing large numbers of servers, available under the EPL (Eclipse Public License), and used for node provisioning and management on the PDSF and Carver systems.
- **Cfengine3**[6]: A popular configuration management software package, licensed under the GPLv3. Cfengine3 and its predecessor, Cfengine2, are heavily used by NERSC.

[3]http://ark.intel.com/products/codename/48991/Jefferson-Pass and http://ark.intel.com/products/codename/48990/Washington-Pass

[4]http://github.com/scanon/chos

[5]http://xcat.sf.net

[6]http://cfengine.com

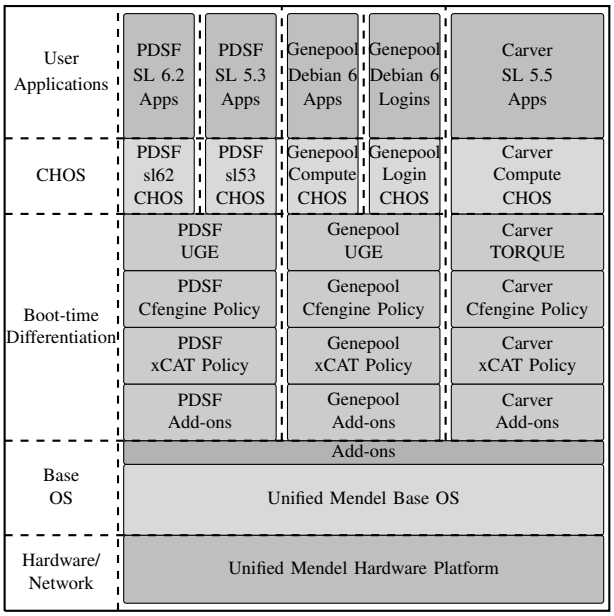


Figure 1. A visual representation of the layered model used on Mendel to accommodate multiple workloads on a unified platform.

- **`image_mgr`** : An internal NERSC tool created to facilitate disciplined management of netboot images. `image_mgr` was created to manage base OS images on the PDSF system, and `image_mgr` leverages multiple existing, powerful tools:
  - FSVS[7], a GPLv3 versioning tool designed to handle the contents of an entire Linux root filesystem using a Subversion repository for backend storage. FSVS is fast enough to handle thousands of small files, including symbolic links, device nodes, and other special files.
  - The xCAT `genimage` and `packimage` utilities

The Mendel system uses a single xCAT netboot image, constructed with `image_mgr`, as the base OS on all compute, service, and login nodes. The default configuration of the base OS image consists of secure and sensible default settings that are not tuned for any particular parent system. Once each node is booted via PXE into the base OS image, node differentiation is accomplished using a set of xCAT postscripts and Cfengine rules. This process brings the node into compliance with the configuration requirements specific to its assigned parent system and role. Users interact with the node exclusively through one or more customized CHOS environments that match the environments of the assigned parent system.

V. Base OS Layer

The Base OS layer provides a familiar, flexible, and minimal software layer on the Mendel hardware platform.

[7]http://fsvs.tigris.org

The responsibilities of the base OS layer include providing a consistent and well-performing set of core system software, handling the IP configuration of the management Ethernet network interface, and providing a set of reasonable and secure default settings for essential network services, such as SSH, GPFS, the portmapper, and the NFS client.

A. Design Considerations

Design decisions for the base OS were determined by several major requirements:

- The base OS had to support the CHOS utility and all the CHOS user environments. This necessitated the choice of Linux as the operating system.
- The base OS should differ as little as possible from a well-tested, commonly used platform familiar to software and hardware vendors, in order to provide a supportable configuration for IBM GPFS and Mellanox OFED. This necessitated the use of a glibc-based userland[8], and also motivated the use of a nearly unmodified set of Scientific Linux RPMs.
- The base OS had to provide an environment familiar to NERSC systems analysts, and provide general-purpose troubleshooting tools such as `strace` and `tcpdump`, which also motivated the choice of Scientific Linux.
- The Mellanox OFED stack should either be included in the base OS, or easily integrated at boot time as an add-on.

Provided that all these requirements are satisfied, the design of the base OS could follow maintainability-focused architectural choices. Consequently, the following advantageous practices were implemented:

- The image was optimized for small size. The uncompressed core image is 260MB, and the Mellanox OFED packages add an additional 90MB.
- The contents of the base OS image are intended to remain unchanged after the post-boot node differentiation process is completed. Consequently, the `yum` package, its dependencies, and the metadata stored in the RPM database could all be removed. While "hot-patching" the image is possible, this practice is intended to only be used for applying critical security changes. Changes to the base OS image should be implemented through a complete image rebuild.

B. Base OS Image Management Approach

The traditional method of building images on midrange systems involves a combination of robust platform-management tools and manual, iterative changes. Such an image management process on a cluster managed with xCAT would begin with the creation of a basic image with the xCAT `genimage` utility. Subsequently, system administrators would modify the contents of the generated image to bring it into accordance with system policy, and then use the xCAT `packimage` utility to prepare it for booting. Over time, the image directory would be manually modified many times.

This process enables rapid changes, but also carries multiple disadvantages. It provides no change management or change logging, unless the modifying system administrator manually maintains a detailed changelog. Even then, the full scope of the changes, including exactly which files were modified and exactly how they were modified, are not known unless extraordinarily detailed changelogs are kept. Additionally, no programmatic method exists to recreate the image or revert changes. This situation leaves system administrators without a good understanding of how the image has changed over time.

Consequently, performing several common maintenance tasks becomes difficult. Upgrading the image to a new minor OS release, for example, will require a new `genimage` invocation followed by manual migration of all system-specific changes. Subsequently, system administrators must modify the new image to meet system policy without exhaustive knowledge of the exact changes required to do so. Tracing which set of changes may have introduced a problem, determining when the changes were introduced and by whom, and rolling back those changes, all become time-consuming tasks.

Alternatively, system administrators can choose to rebuild the image every time it must be modified by running the `genimage` utility and then applying system-specific changes. However, this increases the time and effort required to introduce changes, resulting in loss of flexibility. The `image_mgr` utility was written to lessen the inconvenience of this approach, allowing for full rebuilds while still enabling reasonably quick changes.

C. Base OS Image Management Utility

The `image_mgr` utility provides a standardized interface for image creation, manipulation, analysis, and rollback. `image_mgr` automates image rebuilds from original RPMs, and every image change is accompanied with a full rebuild to ensure consistency and reproducibility. System administrators implement system-specific changes to the image by directly modifying the `image_mgr` script. `image_mgr` also provides a size reporting mechanism to measure the impact of changes on the size of the image.

The `image_mgr` model separates the image into two parts: the `core image`, which is active on all nodes, and `add-on packs`, which are sets of files that can be activated at boot time on subsets of nodes via xCAT postscripts.

To minimize the size of the core image, software sets of significant size which are neither needed nor desired on all

[8] A project to determine if GPFS and Mellanox OFED could be made functional under a minimal Busybox and uClibc-based userland environment was not undertaken, but is potentially useful.

node types are packaged as `add-ons` . Good candidates for `add-ons` are software packages that fulfill niche use cases and are either too large to integrate into the core image, or require disruptive changes to core system software packages. Examples of `add-ons` built at NERSC are CernVM-FS[9], Dell OMSA, KVM, Mellanox OFED, and the OSG stack.

In addition to maintainability improvements, this model enables the use of a *single boot image* across a multi-purpose cluster. The same minimal core OS image can be used for compute nodes, login nodes, and service nodes of all parent clusters. All customizations to accommodate the parent clusters' specific needs are handled at the Node Differentiation layer.

The `image_mgr` script creates well-defined interfaces for image building and management. The `image_mgr` utility exposes a set of subcommands for image creation and management: `create`, `tag`, `list-tags`, and `pack`. The `create` subcommand builds an image with `genimage`, applies a set of site-specific customization shell functions, and then commits the resulting image to the "trunk" area of a Subversion repository with FSVS, as shown in Figure 2. Any revision of the trunk can be "tagged" with the `tag` subcommand as a potential production release. The `pack` subcommand checks out any tag and then uses the xCAT `packimage` utility to prepare that particular tag as the production base OS image.

The `create` subcommand creates a new image and checks it into the appropriate trunk area of the FSVS repository.

```
# image_mgr create -p mendel-core.
    prod -o SL6.3 -a x86_64 -m "Test
    build" -u user
```

Once a desired build is achieved, the `tag` subcommand can mark the current trunk as a potential production image.

```
# image_mgr tag -p mendel-core.prod
    -o SL6.3 -a x86_64 -u user1
```

```
# image_mgr list-tags -p mendel-core
    .prod -o SL6.3 -a x86_64
2013-03-01-14-13-45-RELEASE-by-user
2013-02-27-11-10-02-RELEASE-by-user2
...
```

The `pack` subcommand can mark any tag as the production image, and use the xCAT `packimage` utility to transform the image into a bootable state.

[9]CernVM-FS, often shortened to CVMFS, is a FUSE-based caching HTTP filesystem used extensively by the ATLAS project on the PDSF system. http://cernvm.cern.ch/portal/startcvmfs

```
# image_mgr pack -p mendel-core.prod
    -o SL6.3 -a x86_64 -t
    2013-03-01-14-13-45-RELEASE-by-
    user1
```

VI. Node Differentiation Layer

The Node Differentiation layer is responsible for bringing a node that has been booted into the Base OS image into compliance with the parent system's configuration requirements. Mendel provides three differentiation methods at this layer: system-specific `add-ons` to the Base OS image, xCAT postscripts, and Cfengine rules.

The structured language of Cfengine rules makes them the preferred method for describing differentiation requirements. Consequently, Cfengine not only maintains promises on all nodes, but also forms a core component of the node boot process. This significantly extends the use of Cfengine beyond its traditional boundaries. Cfengine rules install and maintain system-specific configuration files, such as authorization and

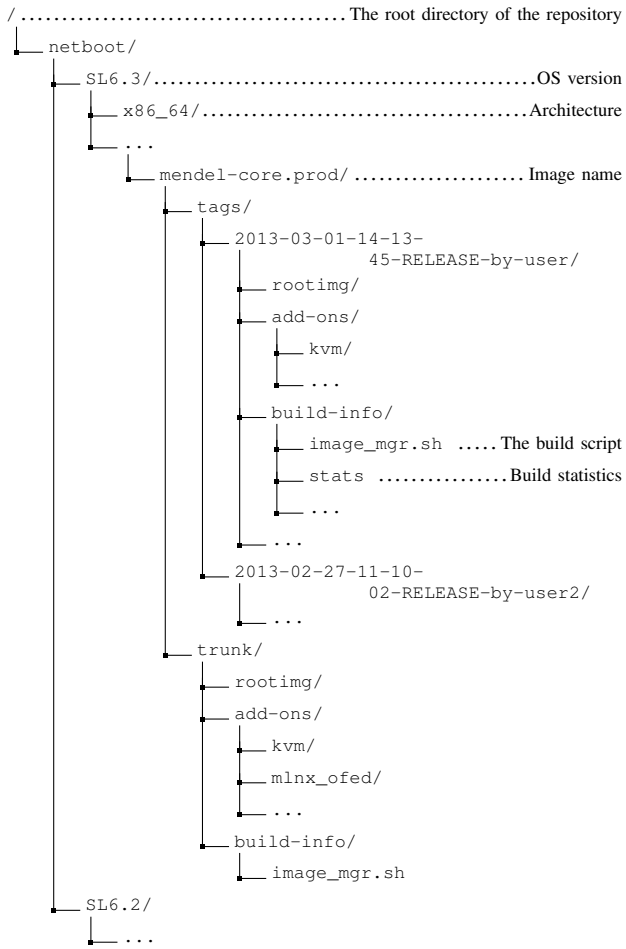


Figure 2. An abbreviated layout of a `FSVS` repository for netboot images managed by the `image_mgr` utility.

authentication configuration (such as `nslcd.conf`) and `limits.conf`.

Policy management cases not easily implemented with Cfengine rules are handled with xCAT postscripts. These postscripts perform functions such as mounting filesystems from local disks, mounting appropriate GPFS filesystems, setting up network interface IP aliases for participation in a load-balancer pool, and verifying that a node's firmware settings and disk layout are consistent with parent system policy.

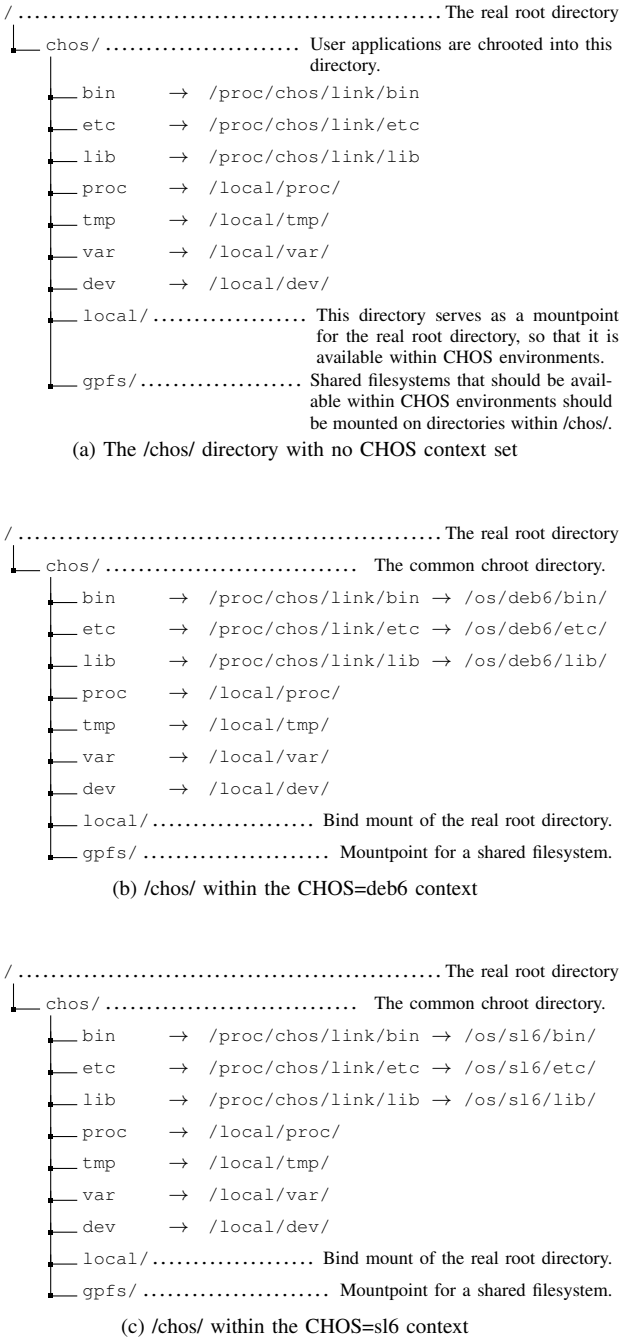
The third method, `add-ons` , are intended for use only when xCAT postscripts and Cfengine rules are insufficient. An `add-on` can be used to add software required by only one parent system. For example, an `add-on` provides the CernVM-FS software package to Mendel nodes belonging to the PDSF parent system.

Once the Node Differentiation process is complete, the appropriate batch execution daemon is launched, and parent system jobs can begin running inside CHOS environments.

VII. CHOS Layer

A 2004 paper [1], by Shane Canon and Cary Whitney of NERSC, describe the design and implementation of the CHOS software package, which combines a Linux kernel module, a PAM module, and batch system integration code to concurrently and seamlessly support multiple Linux userland environments on a single Linux system. The CHOS utility was developed to provide customized computing environments to the diverse projects that share the PDSF system. These projects often rely on application stacks which are only certified or tested on a single Linux distribution release. On PDSF, CHOS has been in continuous production use since 2004, and currently supports SL 6.2 and 5.3 environments on a common SL 6.2 base system. Since its introduction on PDSF, CHOS has served as a versatile and robust method to provide customized computational environments, allowing administrators to support multiple projects while maintaining a minimal base OS image.

The CHOS kernel module manages a contextual symbolic link at `/proc/chos/link`, and maintains a mapping between process IDs and `/proc/chos/link` link targets. The link target is dependent on the CHOS context of the dereferencing userland process, and points to the root directory of the selected CHOS environment. Additionally, the kernel module provides a userland interface through which unprivileged users may set the target of `/proc/chos/link` from among a set of allowed targets managed by the system administrators, and request to be `chroot`ed into the `/chos/` directory . The `/chos/` directory is managed by the system administrator, and contains symbolic links pointing through `/proc/chos/link`, a bind mount of the real root filesystem, and mounts of all shared filesystems. The construction of the `/chos/` directory allows for the maintenance of only a single `chroot` target directory which can assume multiple personalities within different CHOS contexts. Figure VII shows the appearance of an example `/chos/` directory in two different CHOS contexts.



(a) The /chos/ directory with no CHOS context set

(b) /chos/ within the CHOS=deb6 context

(c) /chos/ within the CHOS=sl6 context

Figure 3. Abbreviated `/chos/` directory layout showing the effect of the contextual CHOS symbolic link within different CHOS environments. Figure 3a shows the `/chos` directory when no CHOS context is set. Figure 3b and figure 3c show the contextual appearance of the `/chos/` directory for processes in the `deb6` and `sl6` environments, respectively.

Users may switch among CHOS environments by invoking the `chos` userland utility; however, most users

enter CHOS environments through one of two transparent methods: `pam_chos` or the CHOS batch system integration mechanisms. `pam_chos` is a PAM module which implements PAM session management and places the user into a CHOS environment. `pam_chos` determines the appropriate CHOS environment either through the user-controlled `$HOME/.chos` file, or through the `default` environment from the `/etc/chos` file, and is generally invoked during SSH session establishment. These methods provide transparent integration to all users, and additional control to advanced users.

CHOS provides separation between the user compute environment and the base OS environment. In nearly all cases, interactive user sessions and batch jobs are immediately chrooted into `/chos/`, and users rarely interact directly with the base OS. As a result, system administrators can make significant modifications to the base OS without disrupting the experience of users on the system. Similarly, a user compute environment can be modified without affecting other environments or the base OS. Creating a new user compute environment based on the newest Fedora, SL, or Debian release is possible with no significant changes to the base OS. Additionally, since users can easily switch among environments, users can test and port code to new environments while concurrently running production code in a certified environment.

Based on the success of this approach on PDSF, NERSC chose to use the CHOS utility on Mendel to provide user environments to PDSF, Carver, Genepool, and NX users, on top of the unified Mendel base OS.

CHOS had been used with great success to handle the diverse, but exclusively serial and UGE-managed, workloads on PDSF. Consequently, supporting the PDSF and NX environments on Mendel required no significant changes. Supporting the Carver and Genepool workloads, however, raised new challenges:

- The Genepool and Carver batch environments (UGE and TORQUE+Moab, respectively) handle multi-slot and multi-node jobs, in contrast to the exclusively serial workload on the PDSF system. The CHOS batch system integration code needed to be enhanced to integrate with the multi-slot job launching process.
- Multiple job submission workflows on the Genepool system involve passing a complex set of command-line arguments, including shell redirection characters, to the UGE `qsub` job submission utility. The CHOS UGE `starter_method` needed extensive enhancements to enable seamless handling of these cases. Additionally, the `qlogin` UGE utility, used to establish interactive batch sessions, does not use the `starter_method` by design, and was reimplemented as a wrapper script around the `qrsh` utility.

Table I shows the CHOS environments in use on Mendel.

Name	OS	Purpose
sl62	SL6.2	Production PDSF SL6.2 computational environment
sl53	SL5.3	Production PDSF SL5.3 computational environment
sl5carver	SL5.5	Production Carver computational environment
sl6nx	SL6.3	Production environment for the NX service, featuring commercial NX server software and a full-featured KDE4 desktop stack
deb6gp	Debian 6	Production Genepool computational environment
deb6gplogin	Debian 6	Production Genepool login node environment
deb6dev	Debian 6	Full-featured Debian 6 environment reserved for staff testing and benchmarking
sl6dev	SL6.3	Full-featured SL6.3 environment reserved for staff testing and benchmarking

Table I
CHOS ENVIRONMENTS IN USE ON MENDEL

To support the Genepool system, two CHOS environments, both based on Debian 6, were created: `deb6gp` and `deb6gplogin`, based on the existing compute and login node images, respectively.

To support the Carver system, the `sl5carver` CHOS environment, based on SL5.5, was created, based on the existing Carver compute image.

To support the PDSF system, the existing `sl53` and `sl62` CHOS environments were used verbatim.

To support the NX servers, the `sl6nx` environment was created.

A. CHOS Batch System Integration

Integration of parallel jobs with TORQUE and UGE can be performed using one of two methods. The first method, termed "loose integration" by UGE, involves providing the first task of the job with a list of all slot locations allocated to that job. The job itself is responsible for launching processes within those slots. For example, a TORQUE job can be passed a list of slots via the `$PBS_NODEFILE` environment variable, and can then launch processes within these slots via SSH.

The second method, termed "tight integration" by UGE, utilizes facilities built into the batch system to launch all job processes. Both TORQUE and UGE provide interfaces to expose this functionality. While applications must be modified to support a batch system's tight integration facilities, tight integration allows for far greater control over job process launches and management. OpenMPI's module framework architecture provides `ras` (Resource Allocation System) frameworks to enable integration with TORQUE's `TM` (Task Manager) API and the UGE `qrsh` mechanism.

A successful CHOS-integrated job launch requires all job processes on all nodes to be launched within the CHOS environment. For serial jobs, this is accomplished by configuring a `starter_method` (UGE) or `job_starter` (TORQUE) that launches the job within the appropriate CHOS environment.

CHOS integration for common loose integration cases is handled by the `pam_chos` PAM module, as the SSH server daemons are configured to utilize the PAM stack for session management.

Support for the tight integration case with UGE was also provided through existing `starter_method` functionality. However, tests with TORQUE 4.1.4 indicated that the `job_launcher` was only used to handle the launch of the first process of each job, and not for subsequent process launches through TM. To remedy this, NERSC developed a patch to TORQUE's `pbs_mom` which alters the behavior of `TM`'s process spawning to wrap all process launches with the `job_launcher`. The patch has been accepted, with minor changes, to the TORQUE 4.1-dev branch.

VIII. Future Work

A. CHOS Development

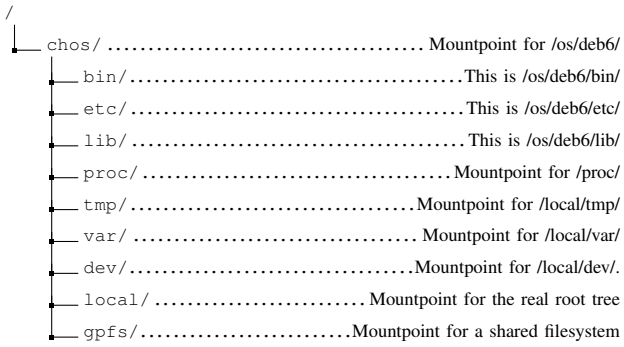
The fundamental design of CHOS has remained unchanged since its 2004 release. Since then, the rapid pace of Linux kernel development has required NERSC staff to port the CHOS kernel module to new Linux kernel versions. This porting effort has remained confined to the set of kernel versions used in production with CHOS by NERSC. The CHOS kernel module currently supports the x86_64 EL5 (2.6.18) and EL6 (2.6.32) kernels released with versions 5 and 6 of Scientific Linux, respectively.

Additionally, until the release of Scientific Linux 6, the CHOS kernel module could be loaded into unmodified kernels from the Scientific Linux kernel release families. The Scientific Linux 6 kernel family, however, is built with the `CONFIG_DEBUG_RODATA` Kconfig option set. This option sets the text segment of kernel memory read-only, which disrupts the CHOS kernel module loading process. Consequently, the EL6 kernel must be rebuilt with this option disabled for CHOS to function, which removes a useful security enhancement and increases the system administration burden on CHOS-enabled systems.

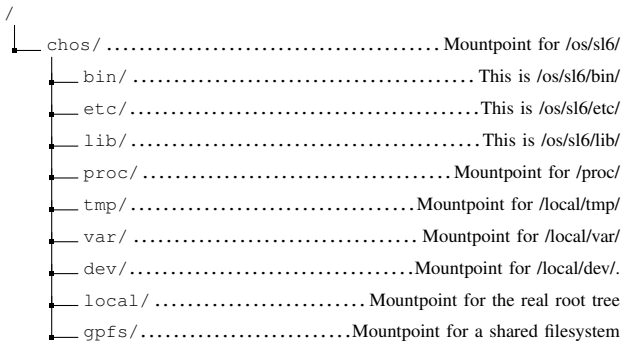
To remedy this situation, investigation is underway to determine if the Namespaces[10] features available in recent Linux kernel versions could serve to replace the `/proc/chos/link` contextual symbolic link and consequently reduce or remove entirely the need for the CHOS kernel module. An experimental branch of CHOS[11], removes the kernel module and instead relies on Linux Mount Namespaces to construct contextual sets of bind mounts within the `/chos/` directory.

[10] http://lxc.sourceforge.net/index.php/about/kernel-namespaces/

[11] http://github.com/scanon/chos/tree/ns



(a) /chos/ under the experimental Mount Namespaces CHOS branch within the CHOS=deb6 context



(b) /chos/ under the experimental Mount Namespaces CHOS branch within the CHOS=sl6 context

Figure 4. Abbreviated `/chos/` directory layout under the experimental Mount Namespaces CHOS branch. Figure 4a and Figure 4b show the appearance of this directory within the `deb6` and `sl6` CHOS contexts, respectively.

The responsibilities of handling bind mounts, placing users into the `/chos/` directory with `pivot_root`, and handling environment teardown, which involves a `pivot_root` back into the real root filesystem and unmounts of all the contextual bind mounts, are moved into the `chos` userland utility.

B. `image_mgr` Development

The `image_mgr` utility provides effective management of a single Base OS image. However, changing the image requires modifying the `image_mgr` utility itself. When `image_mgr` was used only on the PDSF system, this requirement did not create maintainability problems. However, `image_mgr` is now used on PDSF and Mendel. Consequently, two divergent versions of `image_mgr` are now in use. To expand the use of `image_mgr` to more systems, the script requires architectural modifications to support handling multiple base OS images.

C. CHOS Environment Management

While the `image_mgr` utility provides a structured interface for managing the base OS image, no such facility

exists for creating and altering the CHOS environments. The `image_mgr` interface could be adapted to provide such a facility, bringing automatic change logging, rollback, and FSVS versioning to CHOS environment management.

IX. Conclusion

The layered Mendel combined cluster model developed at NERSC provides a framework through which the workloads of multiple Linux computational systems and support servers, which differ significantly in design, implementation, and purpose, can be processed on a unified hardware and software platform. While this approach introduces additional complexity, it provides separation between the software environments used for user applications and those used for system management, allowing all such environments to be architected exclusively for their intended uses. Additionally, Mendel nodes can be easily reassigned to a different parent system by changing xCAT and Cfengine, modifying local disk partitions and firmware settings, and rebooting the affected nodes. The resulting platform is sufficiently flexible to adapt to changing user needs while minimizing user impact and systems management overhead.

Acknowledgment

Substantial credit for the successful deployment of the Mendel platform belongs to several members of the NERSC staff:

- Doug Jacobsen designed and developed the enhancements to the Genepool UGE `starter_method` to accommodate complex `qsub` invocations, and the reimplementation of `qlogin` in terms of the `qrsh` utility.
- Nick Cardo and Iwona Sakrejda provided constructive feedback regarding the implementation of the `image_mgr` utility, and concrete suggestions for improving the utility.
- Shane Canon, the initial CHOS developer, provided significant guidance during the CHOS deployment on Mendel.
- Zhengji Zhao performed many of the initial software tests on the Mendel platform. These tests were instrumental in identifying several key configuration deficiencies very early in the deployment phase.
- Brent Draney, Damian Hazen, and Jason Lee provided extensive assistance integrating Mendel into the NERSC site network.

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] R. S. Canon and C. Whitney. (2004) CHOS, a method for concurrently supporting multiple operating systems. [Online]. Available: http://indico.cern.ch/getFile.py/access?contribId=476&sessionId=10&resId=1&materialId=paper&confId=0