Lonestar 5: Customizing the Cray XC40 Software Environment

C. Proctor, D. Gignac, R. McLay, S. Liu, D. James, T. Minyard, D. Stanzione
Texas Advanced Computing Center
University of Texas at Austin
Austin, Texas, USA
{*cproctor, dgignac, mclay, siliu, djames, minyard, dan*}*@tacc.utexas.edu*

***Abstract*—Lonestar 5, a 30,000 core, 1.2 petaFLOP Cray XC40, entered production at the Texas Advanced Computing Center (TACC) on January 12, 2016. Customized to meet the needs of TACC's diverse computational research community, Lonestar 5 provides each user a choice between two alternative, independent configurations that are robust, mature, and proven: an environment based on that delivered by Cray, and a second, highly customized environment that mirrors Stampede, Lonestar 4, and other TACC clusters.**

This paper describes our experiences preparing Lonestar 5 for production and transitioning our users from existing resources. It focuses on unique features of the system, especially customizations related to administration (e.g. hierarchical software stack; secure virtual login nodes) and the user environment (e.g. consistent, full Linux environment across batch, interactive compute, and login sessions). We motivate our choices by highlighting some of the particular needs of our research community.

***Keywords*-Texas Advanced Computing Center; Lonestar 5; Cray; XC40; Software Environment**

I. Introduction

January 2016 marked the advent of a fifth generation of the Lonestar series of supercomputers deployed at the University of Texas at Austin-based Texas Advanced Computing Center (TACC). Over 3500 active users began the transition from Lonestar 4, a 22,000 core Dell CentOS Linux cluster, to Lonestar 5, TACC's new 30,000 core, 1.2 petaFLOP Cray XC40 supercomputer. Serving as a unique and powerful computational resource for the University of Texas research community, as well as public institutions and other partners across the state, Lonestar 5 supports a diverse range of research interests across dozens of fields of science. This spectrum of users drives the demand for intuitive, adaptive, and efficient environments that work for the researcher.

Our paper will offer a look behind the scenes at workflow needs that helped to motivate the choices behind the crafted user environment custom-built to satisfy our eclectic user base. Parts of the design, construction, and administration of the Lonestar 5 computing environment are illustrated to emphasize the functionality and accessibility that has become indispensable on other TACC cluster systems. We also discuss our experience during both pre-production and the first months of production operation.

This system offers users a choice between two robust and vetted computing environments: 1.) a variation of the Cray environment and 2.) a highly customized TACC environment that tailors other vendor-independent environments familiar to the open-science community. The customized TACC environment leverages proven Cray infrastructure in ways that are often invisible to the user. This is executed in a manner to relieve users from the burden of needing to worry about the implementation, facilitating existing TACC-supported researchers to smoothly migrate workflows to Lonestar 5.

At the heart of the TACC environment lies a set of customized scripts that govern shell startup behavior. User environments are automatically and efficiently propagated from login sessions to batch and interactive sessions. The result is consistency: the environment that the user sees is the same whether on a login node, in an interactive session, or running via a batch submission. Thanks to a tailored network and firewall rules that allow the Slurm workload manager to communicate directly with all login nodes, users are able to gain access directly to 1252 24-core Haswell compute nodes via ssh connections maintained by prolog/epilog scripts in conjunction with Slurm's Pluggable Authentication Module (PAM) [1]. This allows for a minimum of disruption when transitioning from a development cycle to production-level computation.

The TACC user environment has been honed over several generations of compute systems to provide a scalable and flexible platform to address workflows ranging from massively parallel system-level computation, to serial scripting applications conducting ensemble studies, to visualization of complex and varied data sets. Built upon this environment framework, Lonestar 5 implements a hierarchical software tree managed by TACC's own Lmod module system. This Lua-based implementation of environment modules makes it easy to define and manage application and library dependencies so they are consistent across both the login and compute nodes, and allows users to define their own custom software collections to meet their specific needs.

To support protected data including but not limited to applications and data sets falling under International Traffic in Arms Regulations (ITAR), secure virtual login nodes contained within an image of Cray Development and Login (CDL) employ a multitude of logging and security tools to audit the contents of users' protected files while allowing

access to the same open science resources enjoyed by our broader research communities.

Multiple, portable build environments reside in their own change rooted (chroot) jail. Each is a copy of an up-to-date image of a compute node; support and operations staff use these environments to compile and test staff-supported open science and protected data type applications before full production-level deployment. This helps to ensure a safe and dependable creation of over one hundred staff-supported applications on Lonestar 5 minimizing the risk inherent with the operation and maintenance of globally shared production file systems.

A custom MPI wrapper module connected to Cray's MPICH library implementation is available in the TACC environment. This tool provides functionality and familiarity while being able to maintain the scalability and performance inherent to Cray MPICH. Combining Cray's User-Level Generic Network Interface (uGNI) with the efforts of the MPICH Application Binary Interface (ABI) Compatibility Initiative, we have been able to test multiple MPI libraries across the Aries network [2]. Coupled with Lmod and the RPM build environments, we have been able to sample in short order a variety of off-the-shelf parallel implementations to potentially be integrated in future staff-supported software releases.

Lonestar 5 renews TACC's relationship with Cray, one of the leading companies in the high performance computing (HPC) sector. The hardware, tools, applications, and best practices of Cray and TACC together allow our staff to dynamically respond, assess, and address the needs of user communities that are growing in number, depth, and breadth at record pace. This paper identifies the best of these key features and implementation details that make us proud to welcome Lonestar 5 into the TACC supercomputing family.

II. Lonestar 5 Environment

Propagating a consistent, full Linux environment from the moment a user logs in through each and every job submitted stands as a keystone spanning all major TACC HPC systems. Many pieces work in concert to provide a powerful and intuitive experience that has become essential for our research communities.

The foundation of a typical TACC cluster environment is built upon customizations of shell startup behavior. These scripts define a persistent core collection of functions and aliases that provide the ability to easily navigate file systems and applications as well as display system and user status information for any login session. This translates to a shell environment that other tools and applications can then rely upon for providing a reproducible start point. Section III delves into more detail about the customizations created to work in harmony with the existing Cray infrastructure.

Up front, users are presented with a familiar and standardized set of four network-mounted file systems: */home*, */work*, */scratch*, and */corral*. While */home* and */scratch* are single-system NFS and Lustre file systems respectively, */work* (Lustre) and */corral* (NFS) are shared across all major TACC HPC resources. Users are provided a collection of aliases and environment variables that allow for fluid navigation across these file systems.

As on other TACC resources, users gain access to Lonestar 5 through a set of login nodes. For the XC40, the login nodes are classified as External Services Login (esLogin) nodes. They will be referred to as "login nodes" throughout the remainder of this paper. These shared resource machines serve as a location for file transfer, compilation, and job submission. The environment present on these login nodes is also present on the compute nodes. In addition to the shell startup behavior modifications, to achieve this uniformity, network customizations, discussed in Section IV, allowed for a modified build of Slurm in native mode, discussed in Section VII, to communicate and negotiate user ssh traffic directly to and from compute nodes without the use of intermediary batch system management (MOM) service nodes or wrapper applications.

The Lmod module system is utilized to manage the software stack, both for Cray-provided applications with their TCL-based modulefiles as well as for TACC staff-supported packages. Leveraging the startup behavior described in Section III, Lmod (Section V) has been heavily integrated into customized application support (Section VI) best practices.

Up to this point, the components mentioned in this section are present, in some form, regardless of what environment users choose to utilize. Upon login, users default into the "TACC environment". The changes made on the XC40 to support this environment are the main focus of this paper. Where appropriate, asides will be provided to help clarify specific details about any changes or important points with regard the Cray environment.

Users may transition to the "Cray environment" by issuing the command `cray_env` and, as prompted, log out of their current Lonestar 5 terminal sessions to login in once again. At this stage, the user is presented with an environment that resembles the unmodified, off-the-shelf, Cray-provided environment. Users accustomed to working on a Cray system running native Slurm would notice only a handful of changes, none of which would necessitate a significant alteration in workflow on their part. To transition back to the TACC environment, users may issue the command, `tacc_env`, log out of all current Lonestar 5 sessions, and log in once more.

III. Shell Startup Behavior

With every log in, script executed, or file transferred, users and their programs negotiate shells that interact with the operating system. Customizations, typically at the user level, are commonplace, usually through a set of predefined files

that are called upon in a well-defined hierarchical nature at specified event points, such as log in or subshell execution.

Both TACC and Cray employ site-wide shell startup modifications to the common shells (bash, (t)csh, zsh, etc.) provided by SUSE. System-level shell startup behavior begins in the */etc* directory in which specific scripts initialize the content found in the */etc/profile.d* directory in alphanumeric order.

To provide users with a choice of either Cray or TACC environment, it was necessary to construct customized shell startup behavior in both cases. To keep environments distinct, the presence of a particular file in a user's home directory signaled startup scripts which environment to initiate. The creation and destruction of this file, *~/.use_cray_modules*, is managed by calls to the `cray_env` and `tacc_env` commands respectively.

This environment choice was provided for the login and compute nodes only. Service nodes integral to the Aries network fabric need not have a modified environment given users do not run there. To distinguish these nodes from others, the startup scripts would query the system kernel via a call to `uname -r`. Login nodes return `<version>-default` where `<version>` is the Linux kernel version. Compute nodes return `<version>-cray_ari_c` and service nodes return `<version>-cray_ari_s` where **c** and **s** stand for compute and service respectively.

From there, an environment variable, `${__CRAY_STARTUP__}`, is specified if the kernel version is not a compute kernel and not a login kernel. TACC startup scripts are then given a sentinel, where if the variable was defined, the contents of the file are ignored. In two cases, a Cray startup file is also modified to be ignored, one on login nodes (*modules.sh*) and the other on compute nodes (*zz-cray-hss-llm.sh*), to allow for correct TACC environment startup behavior to occur without conflict.

A tool that had previously been developed in-house to help diagnose user's shell startup problems, e.g. a malformed *~/.bashrc* file, proved to be invaluable in first understanding startup behavior and then engineering this relatively simple mechanism to provide distinct environments based on kernel and file triggers. The tool, Shell Startup Debug [3], is able to show a shell's startup path, execution times for each startup file, and also track propagation of environment variables of interest, all from the presence of an environment variable set within a file in a user's home directory. This immensely powerful collection of scripts empower individuals at the user level to track and diagnose startup issues that otherwise would be impossible to see and, at best, tedious and difficult as the root user.

IV. Network Customization

To provide users with a consistent experience across all TACC resources, routes were created to connect the compute nodes residing within the Aries network directly to login nodes. Prior to reconfiguration, login nodes leveraged the Cray-provided `eswrap` module to collect and route specific commands via ssh from the login nodes to a MOM service node. This service node would then reissue the user's commands to utilize compute resources.

All login nodes and the Slurm database (SDB) node were given an additional internal IP address reachable from within the compute private network. This allows direct communication from the login nodes to the SDB node. A connection was made by adding a one 1GigE TP Ethernet cable from the login switch to the SDB node. Slurm traffic, e.g. `sbatch`, `scontrol`, uses this route directly.

To allow ssh communication to the Aries network, all compute node IP addresses were added to the login nodes' */etc/hosts* files. Login nodes were then set up to route this traffic to the boot node. Network address translation (NAT) was configured on the boot node to switch login traffic over to the internal Aries network destined for the appropriate compute node. With the inclusion of Slurm's PAM, only a user requesting nodes through a particular Slurm job may gain exclusive ssh access to their subset of compute nodes.

When a job is launched onto the compute nodes, processes are executed in a change rooted shared-root file system. Slurm's prolog script is initiated and run by the Slurm user before any user processes begin. To gain access to the compute nodes, a temporary RSA host key is generated for the life of the job and an ssh daemon is started, listening on port 6999 with this host key. This is to avoid conflict with the already running ssh daemon outside the chrooted file system. From this point onward, various checks and usage-gathering daemons are initiated before relinquishing the compute nodes over to the user's job.

As a job is completed, Slurm's epilog script is executed. Any remaining user processes are cleaned up and given a chance to exit gracefully. The remaining processes, including the ssh daemon, are terminated before epilog is complete.

V. Lmod Module System

Users of HPC resources often work with specialized software applications that demand the highest levels of performance. This requires resource providers to support a number of different libraries, applications, compilers and MPI distributions. With even a modest set of packages and versions available, the choice of how to clearly and concisely present these tools to the user is crucial. The Environment Modules tool [4], based in TCL, has long served as a solution for managing pieces of the environment such as `PATH` or `LD_LIBRARY_PATH`. When faced with choosing a schema for how to organize packages through their modulefiles, one approach is a hierarchical convention where the currently loaded compiler and MPI distribution govern what modulefiles are available. Modulefile location, as opposed to manually coded logic, dictates swapping behavior when one

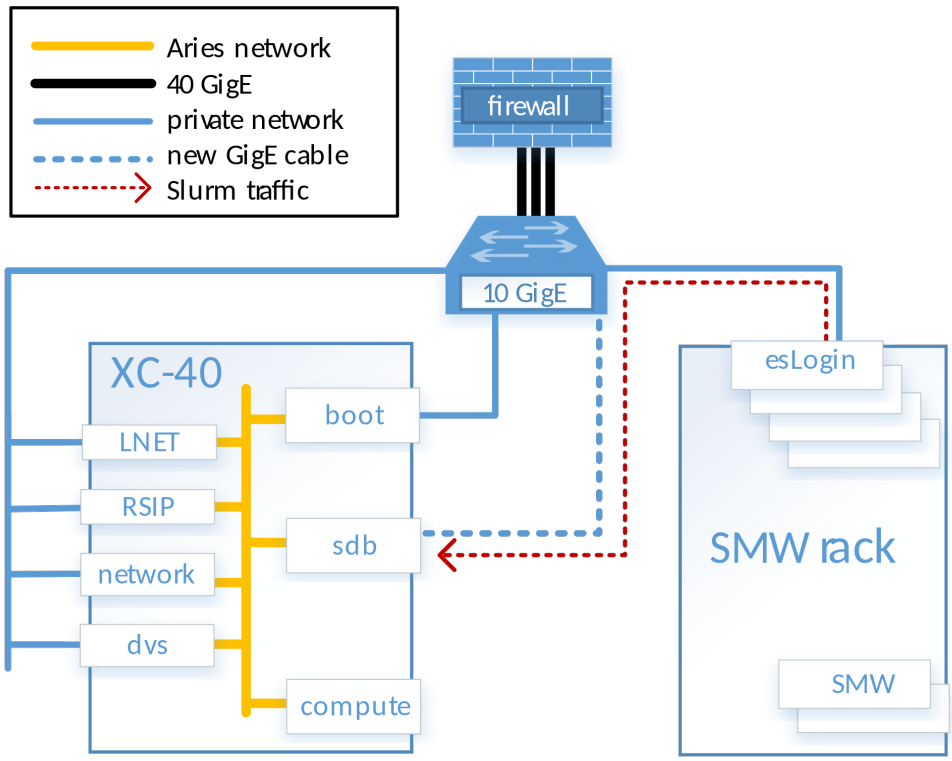

Figure 1. The Lonestar 5 network is configured such that Slurm traffic travels directly to the Slurm database node and also to allow users to ssh directly to compute nodes within the Aries fabric.

package is substituted or removed. This approach inherently helps to prevent package incompatibilities from the user's perspective and simplifies the maintainer's task of tracking dependencies.

At TACC, we implement Lmod [5], [6]; a Lua-based module system that provides several key features that help protect and guide users' choices when navigating potential software combinations. Of the most important features, dependent modules are automatically reloaded when their upstream dependencies change. Moreover, by default, users cannot load more than one module at a time that belong to the same module family or are different versions of the same module.

Given the example in Figure 2, `git`, a version control tool, is considered an independent or "core" package. This is to say that its modulefile may be loaded at any time and is independent of what compiler (Intel, GCC, etc.) and MPI distribution (Cray MPICH, Intel MPI, etc.) modules happen to be loaded. Core packages tend to be built against the GCC compiler that the system uses natively which stands outside of the module hierarchy. In the case where no compiler, and, hence, no MPI distribution modules are loaded, only core package modules will be available.

When a compiler module is loaded, such as Intel 16.0.1 in the example figure, an additional path is prepended to the environment variable `MODULEPATH`. Lmod will use the additional path as an extra branch to search for modulefiles. For the Intel 16 series of compilers, the core-level directory *intel16/modulefiles* is appended to `MODULEPATH`. All packages that are dependent upon Intel 16 at run time would reside inside the *intel16* directory. In this example, upon loading Intel 16, the `boost` library modulefile would then subsequently be on the `MODULEPATH` and would be available to load if the user wishes.

With a particular compiler loaded, typically a set of MPI distribution modules are made available. Similar to the compiler modulefiles, MPI distribution modulefiles will prepend an additional path onto the current `MODULEPATH`. In this example, the directory *cray_mpich7_2/modulefiles* is added when Cray MPICH 7.2.4 is loaded. Packages that are dependent on this particular MPI distribution, e.g. `PETSc`, along with their modulefiles will reside inside the *cray_mpich7_2* directory.

This hierarchical structure ensures that a compatible version of a package is loaded even when compiler or MPI distribution is swapped out. Continuing with the example and assuming that `intel/16.0.1`, `cray_mpich/7.2.4`, and `petsc/3.6` are currently loaded, if, for instance,

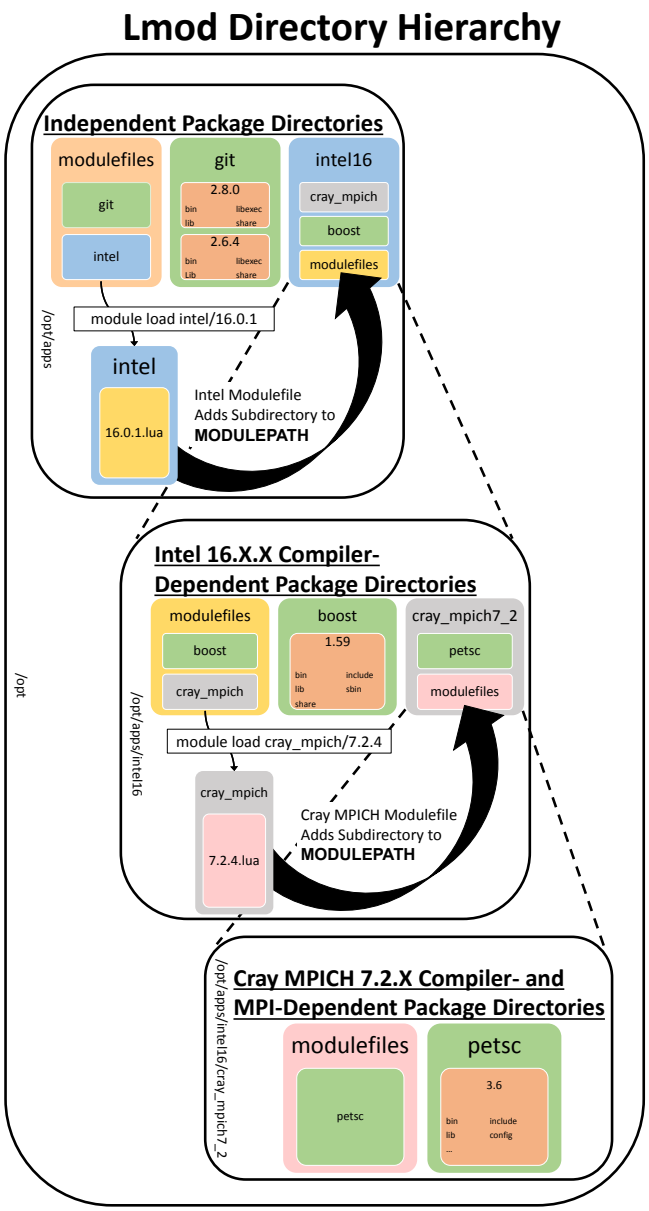

Figure 2. Lmod provides an intuitive, HPC-centric software module hierarchy to coordinate and organize staff-supported packages within the TACC environment.

a user were to issue a `module swap intel/16.0.1 gcc/4.9.3`, Lmod would search for a `gcc/4.9.3` compatible `cray_mpich/7.2.4` and `petsc/3.6`. If found, these package modules would automatically be reloaded to ensure consistency. In the event that one or more dependent package modules do not exist in a branch, the currently loaded dependent module would become inactive.

To help users see what modules are available, the command `module avail` will show all modules that are currently loaded on the `MODULEPATH`. To transcend the current `MODULEPATH`, and see or query all potentially available modules, the command `module spider` can be used. This command can show modules by keyword and version and will provide a dependency list when a single module is queried. Typically, a "spider cache" can be set up, either at user or system level to index results and make spider return immediately.

Generally speaking, different compiler modules need not be loaded simultaneously. Usually, having, for instance, the Intel compiler loaded together with the PGI compiler could lead to unusual behavior. The same could be true of simultaneously loading different MPI distribution modules.

Lmod provides the concept of families as an intuitive and elegant solution for these situations. All compilers can be given a common family tag, usually "comp". By default, Lmod will automatically swap out modules of the same family so that only one is loaded at a time. In the event where the need arises to have more than one module in the same family loaded simultaneously, users have the ability to toggle on expert mode where they are given full control of their environment.

Module collections are another feature provided by Lmod that empower the user to fully customize their workflow environment. In addition to shell startup behavior being consistent across compute and login nodes, so too, are the staff-supported packages that all leverage Lmod. This decision allows a user to craft custom collections of modules that are relevant to their research and to be able to use them anywhere in the system. A default collection can be created by loading the desired modules and issuing the command `module save`. Any time a user starts a new session or issues the command `module restore`, the default collection will be reinstated. Users may also save named collections in the event that they would like to quickly and easily switch between different sets of modules, perhaps for a different workflow. The metadata for these collections is stored in a dot file within the user's home directory.

Lmod provides a multitude of well thought-out enhancements and features over the current Environment Modules package. For this paper, these last noted features give users the ability to quickly and efficiently manipulate their environment in ways not possible with Environment Modules. First, a user may reset the current module stack at any time back to the system-provided default by issuing the `module reset` command. Second, while the `module swap` command is provided, it is in almost all cases unneeded with Lmod. If a user wishes to swap compilers, they need only to load the compiler they want. Lmod automatically takes care of the rest. Third, a user may completely unload their module environment with the `module purge` command. Forth, Lmod supports shorthand notation. For instance, the command `module restore` becomes `ml r`; `module list` becomes `ml`; `module swap intel/16.0.1 gcc/5.2.0` becomes `ml gcc`.

VI. RPM Build Environment

TACC staff build, maintain, and support many dozens of software applications, libraries, and tools. These elements are served and accounted for using RPM Package Manager [7]. An up-to-date, safe, and powerful build environment is needed to facilitate in the creation of these site-customized RPMs. A chroot jail, based on the latest compute node image, provides the opportunity to develop RPMs in a space much like the production environment the RPMs are targeted for. This is accomplished while isolating any changes away from the live system.

The majority of staff-supported packages are served via a shared read-only Data Virtualization Service (DVS) projected NFS mount. This file system, referred to by its mount point, */opt/apps*, is visible from all login and compute nodes. Any change to */opt/apps* is affected immediately throughout the system. This allows for quick and simple management of content served from the master administration node. New packages can be installed on-the-fly without need for maintenance or downtime. This serves as a double-edged sword in the sense that live packages could be removed while in production. The chroot jail environment is purposefully disconnected from the production */opt/apps* to remove the potential of impacting running jobs while RPMs are in development.

An instance of the RPM chroot jail is created on one of the System Management Workstation (SMW) nodes from the current compute image located on the boot node. Content is copied to a staff-accessible subdirectory within */opt/apps* where several files and directory mounts are connected, some bind mounted, to provide a persistent experience between updates. A Bash function is provided to staff with elevated privileges to ensure the consistency and integrity of the jail before transitioning into the change rooted environment. An updated RPM jail is typically created at the end of regularly scheduled maintenance periods after all other compute image changes have been completed.

Multiple instances of RPM chroot jails can be created at different mount points. Specific community jails exist for the development of open science packages and for export controlled packages that fall under ITAR. Staff may also opt to clone an instance of a community jail in the event that a personalized isolated environment is necessary.

Staff using the RPM jail may transition between TACC and Cray provided environments using the same commands as outlined in Section III. While a few packages are maintained for the Cray environment by TACC staff, the majority of the staff-supported software stack is custom built for the TACC environment. At the time of this writing, just under 100 unique modules are provided, many with multiple versions across the Intel and GCC compilers and Cray MPICH MPI distribution. For perpsective, Lonestar 4 served over 225 unique modules at its retirement. In total, over 170 modules are currently provided and maintained. This number will continue to grow as Lonestar 5 prepares to cross its three month deployment mark.

RPM spec files for each package are curated by a number of staff members, many of which use template skeletons to aid in an efficient build process. These spec files are tracked via version control and can be migrated between systems with ease; usually only needing minor updates upon a new system build. A fifteen line nested if-condition included in all spec files controls package and modulefile names

plus paths in the form of RPM macros. These are used to integrate into the Lmod module hierarchy by helping to create modulefiles directly in the RPMs and to standardize RPM naming conventions. This snippet of logic is all that is needed by developers to produce a structured and consistent software stack able to harness Lmod.

VII. Workload Management

Thanks to recent advancements made by SchedMD and Cray, TACC joins a growing number of supercomputing sites operating Cray XC systems with Slurm in "native" mode [8]–[10]. Slurm control and database daemons continue to reside on the Slurm Database (SDB) service node within the Aries fabric. In native mode, Slurm daemons now reside directly on the compute nodes. In this mode, there is no need for the Cray Application Level Placement Scheduler (ALPS) daemon nor the calls to `aprun` that most Cray users would be familiar with. Instead, Slurm's `srun` is used for parallel job launching.

Native Slurm leverages alpscomm, a low-level network interface library, to communicate with the Aries fabric via hardware APIs. Node information including health and state, as well as job launching is managed by Slurm via this communication interface [11]. Native Slurm is installed directly into the Dynamic Shared Objects and Libraries (DSL) shared-root file system that is projected by DVS to the Cray Linux Environment (CLE) compute nodes [12]. Slurm leverages Cray's specialized *libpmi* library, an integral component for MPI jobs to succeed on the Aries fabric.

With routes configured such that login nodes can interact directly with the Slurm database node, Slurm commands do not require any sort of wrapper function to ssh to a MOM service node before queries or job launches. This approach provides a more intuitive and familiar design that also provides the path to which our users may directly ssh to compute nodes.

Clearly this configuration is neither Cray's Cluster Compatibility Mode (CCM) nor its Extreme Scalability Mode (ESM). Our current configuration is probably beyond what Cray would still classify as a Massively Parallel Processing (MPP) supercomputer. Fair enough: ours is a unique, eclectic system that borrows key components from several design philosophies. It is a performant platform with the power and flexibility that TACC-supported researchers have come to expect.

Additional Slurm source code modifications have been employed to aid in job accounting and to help attach a custom filter plugin that allows for additional site-specific settings and imposed resource limits. System administrators can quickly and easily provide user access on a per queue basis, allow specific users to run for extended periods of time, toggle checks for file system health, ensure correct ssh key permissions in a user's home directory, manage system accounting, and disable users who are negatively impacting the system, among many other customized features.

Given this setup, users can issue an `sbatch` command directly to the compute nodes. The TACC Slurm filter processes sanity, accounting, and access checks before initiating a typical Slurm job sequence. Once the job is running, a user may ssh directly to any compute node that their job is running on. While this style is more rare, a representative workflow case could include online job status or performance monitoring. Moreover, this gives consultants the ability to watch a job that a user has initiated for diagnostic purposes.

Long provided as a core piece of functionality on TACC resources, interactive development sessions play a crucial role in the HPC code development cycle. The ability to modify, recompile, and relaunch a job in short order enhances productivity immensely over having to resubmit a Slurm job to busy queues for every change in one's code. These sessions also play a central role as a simple and quick way to have our users in training be able to learn exactly what type of compute environment will be available for both batch and interactive jobs. At the other end of the spectrum, advanced users and consultants can use interactive sessions at the largest scale to interrogate and observe code behaviors that are otherwise incredibly difficult to diagnose when only provided a remote, non-interactive, batch session.

TACC's own tool for interactive access to compute nodes is `idev` [13]. It provides all the functionality needed for a scalable solution that delivers the same environment for interactive development sessions that users see in their batch jobs. Whether providing the twenty-four cores of a single compute node for parallel makefile execution, or harnessing multiple visualization compute nodes for interactive data rendering, `idev` serves as the tool of choice. A user need only to type `idev` to launch an interactive job on one compute node. If more are desired, all standard `sbatch` flags are recognized. As an `idev` job is started, an `sbatch` is launched under the covers. A user's `idev` launch environment is recorded and secure copied into the head compute node's */tmp* directory. Upon shell startup execution, if this file is found, it is sourced to provide a duplicate environment on the compute node. This behavior is true of any ssh sessions to the compute node. Users start their compute session in the same directory, with the same loaded modules, and the same environment variables set.

VIII. MPI

Cray MPICH libraries, with a few adjustments, are provided via an Lmod wrapper module in the TACC environment as the default production-level MPI distribution. Both Intel and GCC compiler versions are used, tested, and maintained to provide the same expected performance typically found within the Cray environment. A few key design

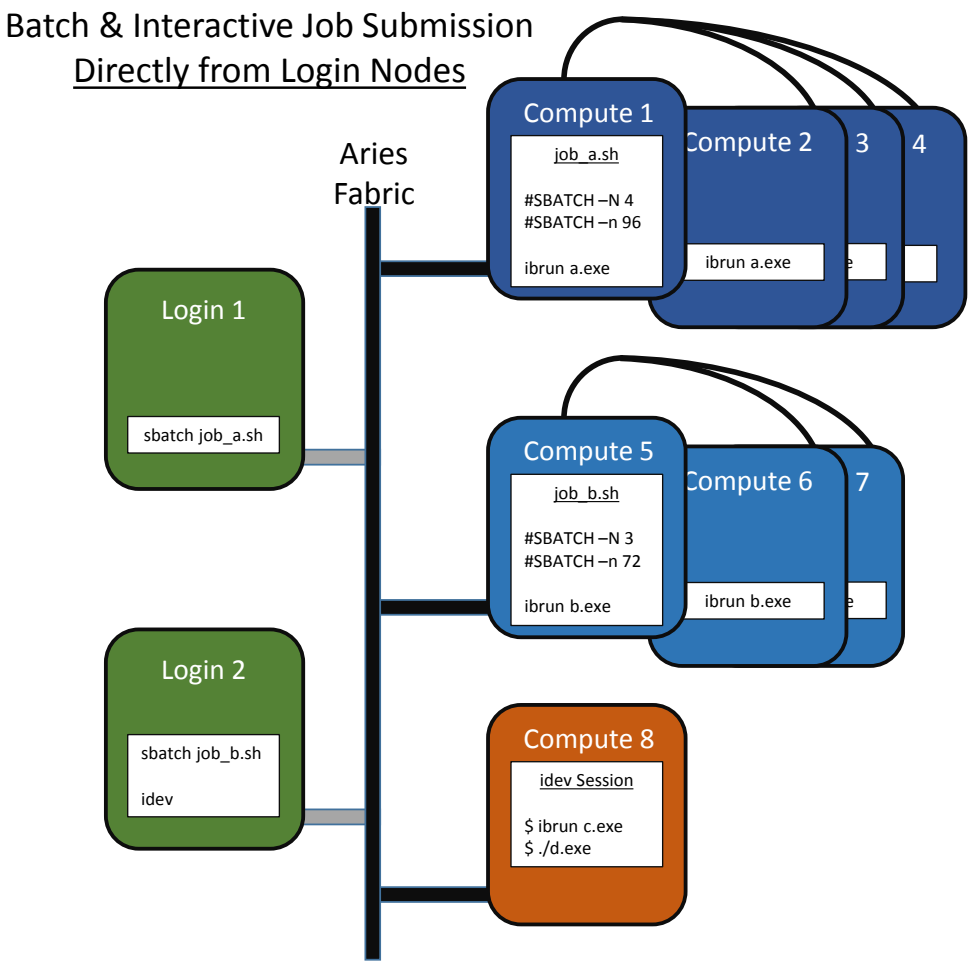

Figure 3. Custom networking and Slurm in native mode provide login nodes direct batch and interactive sessions on the compute nodes.

decisions were made to help functionality and usability by our user community.

The Cray-provided compiler wrapper routines generate a compile-time set of include paths and libraries automatically generated via `pkg-config`. These configurations are built up based upon what modules happen to be loaded at the time of compilation. This approach is inclusive, providing all possible components that a user may need for compilation. The TACC environment is not based upon `pkg-config` directories and metadata. Instead, a smaller number of include paths and libraries are provided to the user in a set of wrapper scripts renamed `mpicc`, `mpicxx`, and `mpif90` to align with MPI distributions provided on other TACC resources. For the applications tested by TACC staff, no discernible differences in performance have been documented.

On TACC resources, an MPI job launcher wrapper script called `ibrun` is provided to simplify a user's experience [14]. The different launchers (`mpirun`, `mpiexec.hydra`, `mpirun_rsh`, `srun`, etc.) all provide different flags and calling options. Moreover, many launchers expect a user-provided host list. Working on a production system, Slurm jobs will in almost all cases be on a differing set of compute nodes than previous runs. The `ibrun` wrapper is a way to abstract this layer and provide a uniform experience regardless of MPI distribution, hardware layout, and host list. On Lonestar 5, the `ibrun` script has been adapted to call `srun` with the correct options based on a user's input arguments.

Inevitably, with such a diverse user base, experiences and knowledge differ about what methods are supported for launching MPI jobs. Through the use of the default TACC Lmod meta-module, we provide a set of wrappers that appear at the beginning of a user's default `PATH`. When called, these wrappers produce a message to inform the user that the current command is not supported and points them to documentation in our user guide. This gentle reminder helps to minimize confusion and is easily super-

seded, by design, by any user-supplied additions supplied to `PATH`. Current wrappers include `mpirun`, `mpiexec`, and `mpiexec.hydra`.

IX. Virtual Logins

The wide diversity of TACC-supported researchers bring to bear many challenges in the form of specialized workflow needs. In some cases, this can include requests for customized login environments. A login node that tries to suit all researchers' needs is difficult to secure and in some cases is not feasible due to the exclusive nature of particular requests. Rather than procure new hardware for each situation, TACC hosts virtual machines (VMs) in-house through a VMware [15] service. In short order, a customized VM for can be spun up for a specified research group. The practice has been carried out for many years and continues to gain in popularity as our user base grows.

To assemble a virtual login (vlogin) node, configuration begins by creating a virtual machine that uses the Cray Development and Login (CDL) install media. The base operating system, security updates, and TACC customized RPMs are installed while a new private network is plumbed from the VMware server to the Lonestar 5 login switch. Once configured, this virtual machine can be cloned and given minor adjustments for another research group or other purpose. Currently, all file systems are mounted via NFS on the VMs including the shared file systems */scratch* and */work*. An instance of the created virtual machine can then be utilized as a virtual login node.

A version of the generic virtual machine described above will soon be deployed into production with several security and monitoring enhancements. The purpose of this virtual machine is to serve researchers whose projects require the use of source code, binaries, and/or data that in some way fall under ITAR. These vlogins require two-factor authentication to gain access and mount a specialized file system that is used to store users' protected ITAR contents.

Core services and file systems are monitored via Nagios, incron, and customized in-house scripts [16], [17]. Events are remotely logged and aggregated via Splunk server infrastructure [18]. Rules are integrated to alert appropriate staff for notable security or environment events. For example, a cron job that runs on the server that houses the ITAR file system is required to update a timestamp on a file within a monitored directory. This heartbeat rule is set up on a Splunk server to send an alert if the timestamp was not able to successfully update. Checks such as these help to ensure that all services and scripts of the monitoring systems are healthy.

Users who access the ITAR file system through a virtual login node also have the ability to read and write contents through Slurm jobs that run on the same compute nodes that open-science researchers access via regular login nodes. Logic in the Slurm prolog and epilog scripts are used to identify specific users, queues, and accounting projects that are authorized to access the ITAR file system. The file system, served by DVS, is dynamically mounted at the beginning of the job and unmounted at the end. In the event that compute node has trouble mounting or unmounting the ITAR file system, the node is automatically disabled and system administrators are notified.

X. Conclusion

When it comes to the crucial components needed to successfully provision a supercomputer, the devil is always in the details. It is easy for us as authors, and you as readers, to be distracted by the many facets and miss the big picture. In the end, three principles informed our software design for Lonestar 5:

- **Familiarity**: Anyone who uses TACC resources should find themselves in an environment that provides the same recognizable look and feel. Those who use both Stampede and Lonestar 5, for example, should have the same experience on both systems. The same should be true for users transitioning from Lonestar 4.
- **Consistency**: No matter where researchers find themselves on the Lonestar 5 system, they see the same full Linux environment delivered with a rich and broad selection of tools, applications, and libraries. This means less distraction fighting the machine and providing more freedom to just get the job done.
- **Flexibility**: TACC's infrastructure supports more than 60 Fields Of Science (FOS), 120 institutions, and 230 research projects. This means we require that our systems provide intuitive platforms that can respond and evolve in ways driven by our users' needs.

Clearly there is more to be done and we don't pretend to have all the answers. From our perspective, preparing Lonestar 5 has and continues to serve as a wonderful opportunity to grow and adapt. This system is, invariably, a different machine from its predecessor. While TACC hosts a variety of HPC resources from many vendors, it has been since Lonestar 1, a 40 GigaFLOP T3E, that a Cray machine was housed in our data center. Challenges lie on the road ahead; some already well known as we define our experiences and refine lessons learned; others, down avenues we have yet to encounter.

It has taken many of hours of hard work but we judge them as successful. With innumerable thanks to individuals, externally and internally, research has continued, by and large, in "business as usual" fashion for the majority of our Lonestar user base. To provide some perspective, Lonestar 4 did not use Slurm as its workload manager. So, some users, transitioning from Lonestar 4 encountered Slurm for the first time. Our support staff report that learning a new workload manager proved to be the hardest part of the transition to Lonestar 5. We take this as evidence of success; we must be doing something right.

REFERENCES

[1] Linux-PAM. (2016) Pluggable Authenication Modules for Linux. http://www.linux-pam.org.

[2] K. Raffenetti. (2016) MPICH ABI Compatibility Initiative. https://www.mpich.org/abi.

[3] R. T. McLay. (2016) Shell Startup Debug: Startup Script Tracer. https://github.com/TACC/ShellStartupDebug.

[4] J. L. Furlani, "Modules: Providing a Flexible User Environment," in *Proceedings of the fifth large installation systems administration conference (LISA V)*, 1991, pp. 141–152.

[5] R. T. McLay. (2016) Lmod: A New Environment Module System. https://lmod.readthedocs.org.

[6] ——. (2016) Lmod: An Environment Module System based on Lua, Reads TCL Modules, Supports a Software Hierarchy. https://github.com/TACC/Lmod.

[7] RPM. (2016) RPM Package Manager. http://rpm.org.

[8] SchedMD LLC. (2016) Slurm Workload Manager. http://slurm.schedmd.com.

[9] ——. (2016) Slurm User and Administrator Guide for Cray Systems Natively. http://slurm.schedmd.com/cray.html.

[10] D. Jacobsen and J. Botts, "Native Slurm on the XC30," in *Proceedings of the The Slurm User Group Meeting*, 2015.

[11] D. Wallace, "Birds of a Feather: Native Slurm on Cray XC30," in *Proceedings of the The International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[12] T. Fly, D. Henseler, and J. Navitsky, "Case Studies in Deploying Cluster Compatibility Mode," *Proceedings of the Cray User Group*, 2012.

[13] K. Milfeld. (2016) idev: (interactive development) User Guide. https://portal.tacc.utexas.edu/software/idev.

[14] K. Schulz and J. Cazes. (2016) Lonestar 5 User Guide: ibrun. https://portal.tacc.utexas.edu/user-guides/lonestar5.

[15] VMWare. (2016) VMWare Home Page. https://www.vmware.com.

[16] Nagios Enterprises LLC. (2016) Nagios: The Industry Standard In IT Infrastructure Monitoring. https://www.nagios.org.

[17] (2016) inotify: Get Your File System Supervised. http://inotify.aiken.cz.

[18] Splunk Inc. (2016) Splunk. http://www.splunk.com.