SLURM. Our Way.

Douglas M. Jacobsen, James F. Botts, and Yun (Helen) He
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

***Abstract*—NERSC recently migrated its primary batch system and workload manager from an ALPS-based solution to SLURM running "natively" on our Cray XC30 and XC40 systems. The driving motivation for making this change is to gain access to features NERSC has long implemented in alternate forms, such as a capacity for running large numbers of serial tasks, and gaining tight user-interface integration with new features of our systems, such BurstBuffers, Shifter, and VTune, while still retaining access to a flexible batch system that can deliver high utilization of our systems. We we have derived successes in all these areas. The largest unexpected impact has been the change in how our staff interacts with the system. Using SLURM as the native workload manager (WLM) has blurred the line between system management and operation. This has been greatly beneficial in the impact our staff can have on system configuration and deployment of new features, becoming a platform for innovation. Conversely, SLURM overlaps in some places with Cray-provided tools that have required some significant adjustment in our system management procedures. On the positive side, this has allowed our engineers to create new ways to performing "rolling" updates of compute nodes while isolating running jobs from negative effects. The interactions between SLURM and the Cray provided tools such as xtprocadmin or the NHC have required increased low level understanding and procedural changes to maintain appropriate resilience.. This transition has required some adjustment from our users, as the "native" SLURM solution uses different methods for launching jobs and tasks. Some former points of contention (e.g. there is now little-to-no reliance on internal login nodes) have disappeared as a result of these changes in batch system architecture. The use of the "native" SLURM allows greater control over how applications are launched and access our resources. With more flexibility, significant user-training has been required, especially in the areas of task/CPU-binding. The adoption of natively running SLURM on our systems has greatly increased our capabilities and broadened staff involvement in detailed system management work. We are still measuring the impact on our user base.**

***Keywords*-slurm; cray; hpc; scheduling; backfill**

I. Introduction

The National Energy Research Scientific Computing center (NERSC) operates two XC-class Cray supercomputer systems. Our XC30, edison, has 134,064 Intel Ivybridge cores over 5,586 compute nodes. Our XC40, cori, is in the first stage of its deployment and currently contains 52,096 Intel Haswell cores over 1,628 compute nodes. In the Summer of 2016, the second phase of its deployment will be augmented with 9,300 Intel Knights Landing many-core processors. NERSC uses these systems to serve the needs of over 7,700 users running a wide variety of hundreds of different scientific codes.

Throughout 2015 and early 2016 NERSC moved from Oakland, CA to a new building at the primary Lawrence Berkeley National Lab site in Berkeley, CA. During this period, the first phase of the cori (Haswell) system was delivered and integrated at the new site during summmer and fall of 2015. We opted to introduce native SLURM as the scheduler and resource manager (collectively Workload Manager) on the cori and edison systems as they became available at the new Berkeley site. From the perspective of our users, cori was introduced with SLURM in October of 2015, and edison was returned to service in January, 2016 with SLURM.

One significant difference between SLURM and other scheduler software available for Cray XC systems, is that, starting with version 15.08, SLURM does not make use of the apbasil interface to Cray's ALPS product. Instead, SLURM uses the so-called "native" interface on the Cray system to allow SLURM to directly manage the hardware resources of the system. In practical terms, this has a number of consequences:

- aprun is no longer used, instead one can directly use "srun" within a batch script
- slurmd, a daemon, is run on each compute node (instead of apinit)
- SLURM makes heavy use of the full Linux environment, requiring slurmd to be exclusively bound (chroot'd) within the /dsl environment for CLE 5.2
- The job batch script is executed on a compute node instead of a service node
- SLURM needs to acquire user information at job start on each compute node, requiring proper configuration of all compute nodes to access site user authentication services to function (e.g., LDAP), which may require RSIP or other means of accessing remote authentication services. Formerly, this external access was only needed by a handful of service nodes acting as intermediaries.

This paper will relate some of our experiences and some of the choices we have made in order to make SLURM a success on our systems. We have worked closely with SchedMD (SLURM vendor) [1] and Cray throughout this

process.

II. SLURM Architecture on Cray XC System

SLURM is comprised of a number of daemons, user executables, libraries, and binary-compiled plugins. All of the major functionality for managing on-node resources and maintaining communication are handled by slurmd, a single daemon. Any job step, such as the batch script or a particular srun, is managed by one slurmstepd daemon per node of the allocation, forked from the associated slurmd on the node. slurmctld, the control daemon, implements all the core functionality of the scheduler and orchestrates the activities of all the slurmds. Any jobs submitted to the system are achieved via direct tcp connection to the slurmcltd. Slurmctld is the central point of communication for both user interaction as well as system management.

At NERSC, we have a number of requirements for our batch system with respect to resource management, availability and accessibility:

- users should be able to submit jobs and query job status at almost any time, regardless of the booted or powered state of the supercomputer
- with a dynamic base of over 7,500 users and over 14,000 unix groups, up-to-date user/group authentication information is needed to be directly accessible on the compute nodes
- we need to update the SLURM configurations without rebooting the system
- we need to update SLURM software (minor revisions) without rebooting the system

To achieve this, SLURM necessarily requires access within the supercomputer to perform node management and job maintenance, and have a strong presence on the user facing external network to enable direct communication with the user front-end processes.

To this end we created a new class of node on our mainframe termed "ctl" (control) nodes. Owing to the fact that internal login nodes are no longer needed (also called "mom" nodes in torque parlance), our former "net" nodes, service nodes which have ethernet connectivity, were no longer required in the formerly necessitated quantities. As a result we run slurmctld on an internal service node with direct ethernet network connectivity to the external service network (also known as the es network).

A. Optimal slurmd and slurmctld communication

To provide access to SLURM even when the supercomputer is down, we run a backup instance of slurmctld on the external service network. Since it cannot communicate directly with the slurmds on the compute nodes, it is not competent to schedule or manage but can accept jobs into the queue. This maximizes batch system availability to a high degree. There is one exception; Cray DataWarp integration with SLURM requires access to the supercomputer DataWarp services, so jobs requiring DataWarp BurstBuffers cannot be submitted to the external slurmctld instance. The slurm.conf needs to specify "no_backup_scheduling" amongst the scheduler parameters to enable this.

To ensure proper routes of communication, it is important that compute nodes access slurmctld via the ipogif (high speed network) interface, not via RSIP. To that end, we explicitly specify NodeAddr within slurm.conf as the ipogif0 address for each compute node definition to ensure that slurmd only binds for listening on the ipogif0 interface. In addition the ControlMachine configuration for slurm needs to match the hostname resolvable as being on the ipogif network; however, the slurm.conf that the primary slurmctld reads must not specify ControlAddr, to ensure that it does listen on all network interfaces.

On the es network, external to the supercomputer, we deploy a slightly modified slurm.conf that specifies a new ControlAddr that points all the slurm commands to use the es-network facing interface on the ctl node. These configurations ensure that slurm daemons and executables have a robust and correct route to communicate with each other.

B. Deployment Issues

As we deployed SLURM we ran into a number of issues, the resolution of which have greatly improved our productivity and may be of interest to others. In particular, we found that having a mechanism for rapidly re-deploying SLURM, the slurmctld in particular was of great value. The value therein is largely because we found that the more we worked with SLURM and SchedMD, we would get, from time to time, patches to solve particular issues (that would appear in future versions). Also, we have a small quantity of our own patches to implement NERSC-specific behavior. Therefore, we constructed an automated build system which would integrate these patches and automatically build SLURM thrice. Our three SLURM installations are:

- the supercomputer installation in /dsl/opt/slurm/<version >(linked to "default")
- the es-network installation in /opt/slurm/<version >(linked to "default")
- an es-network installation for managing the es network resources (this is a separate cluster and is managed independently from the supercomputer installation)

For both of the es-network installations we modified (and upstreamed) slurm to add the "–enable-really-no-cray" configure option which disables all the Cray-specific build requirements. This allows the es-network versions to avoid attempting to link against Cray XC-specific libraries (like libjob, for example). This was necessary because some of the paths used by SLURM to detect that it is being built on a Cray are present in the es network service images.

It is critical that the environments on the eslogins and the supercomputer match in most ways. In particular, the PATH

environment variable should work similarly to ensure that SLURM forwarding of the environment from the eslogin environment is correct. To that end, it is important that the SLURM binaries exist in the same path in both environments. This is why both the supercomputer and es versions of SLURM are installed in /opt/slurm/default (effectively).

1) Avoiding Reboots when Updating SLURM in CLE5.2: Because SLURM is chroot'd to the /dsl environment, it is therefore installed within the shared root, which is served out to the compute nodes via DVS. Early on we found that we could not easily update SLURM, because some subset of the compute nodes would still "see" the previous version of SLURM. The recommendation from Cray was to reboot the supercomputer after updating SLURM. This, as mentioned earlier, was at odds with the need to update SLURM for minor repairs or small modifications of functionality.

We discovered that this was actually caused by the DVS attribute cache on the client (compute-node) side. The attribute cache stores the file ownership, permissions, and the dereferenced values of symlinks for whatever the cache timeout is to prevent excessive metadata lookups. By default this attribute cache timeout is four hours, meaning it could take nodes up to eight hours to see the updated version in the worst case.

To deal with this we modified the fstab of our compute node image to mount /dsl/opt/slurm with an attribute cache timeout of 15s. This means that we can update SLURM, and within 30s all compute nodes will "see" the new version. This does require that the symlink not run through /etc/alternatives though, since /etc will have the same attribute timeout as the other portions of the shared root.

Similarly, we found that there were sometimes improper effects when updating SLURM configuration files in /etc/opt/slurm. These have been mostly rare bugs that only appeared at the scale our cori and edison systems. However, taking advantage of the same DVS mount from earlier, we simply synchronize our configurations into /dsl/opt/slurm/config.<svn revision >and link config.<svn revision >to "etc", such that the final effective path for the slurm configuration is /opt/slurm/etc. Of course, SLURM needs to be configured with a "–sysconfdir=/opt/slurm/etc" for this to work.

These changes allow us to update SLURM and SLURM configuration files in production in a repeatable and scriptable way, which avoids the need to reboot any portion of the system. At the worst, we need to restart some daemons in these situations.

2) salloc, srun, sshare from eslogins: The es-network deployment of SLURM cannot directly communicate with the SLURM compute nodes. Direct communication is required for using srun directly or srun within an salloc allocation. sstat has similar requirements. To allow interactive use of the system, we therefore needed some way to allow salloc, srun and sstat to directly communicate with the compute-node slurmds. Building on the existing concepts of CSCS's ssh-based wrappers and on the Cray's eswrap functionality. We constructed a set of wrappers that would properly conserve user environment (except DISPLAY) and command line options. These wrappers automatically ssh the user from the esnetwork to an internal login node, source their environment, and run their specified command. By wrapping these specific commands and leaving the other SLURM commands (e.g., sbatch, sacct, etc.) alone, we are assured that non-interactive SLURM functionality can continue to run diring outages.

3) Scaling SLURM Up: We first deployed SLURM to cori phase 1, a 1,628 compute node system. Other than balancing the priorities of the system, we have had very few issues running the system.

When we deployed SLURM on edison, however, a 5,586 compute node system, we found that there were some scaling issues that required attention before it would work well. In particular, some of the standard tuning to increase max TCP connections and SYN backlog are useful (sysctl parameters net.core.somaxconn and net.ipv4.tcp_max_syn_backlog should be larger than the quantity of slurmds). Beyond this however, we found that we had to limit the slurm.conf TreeWidth to the cubed root of the number of nodes to reduce communication backlogs within the slurmd communication tree (the usual default is the square root). The SrunPortRange also needed to increase to allow for a sufficient quantity of ports to allow the primary slurmstepd of a job to setup the stdio communication ports. For edison, 2,000 ports per node was found to be sufficient, and that these ports should be non-overlapping with the RSIP port range.

The most critical scaling issue, however, was that the very largest of jobs would frequently fail to run. When jobs requesting a large quantity of tasks (e.g., >60,000 tasks), the job step would exit with a PMI2 failure to initialize message. In the end, we found that this was because of a difference in behavior between srun and aprun. By default, aprun copies executables prior to executing them, whereas srun does not. For most small to medium jobs, the srun behavior is probably fine, if not, "better". However, running a large quantity of ranks directly from the parallel filesystem (lustre, DVS, whatever) would fail because the filesystem could not deliver the executable at that level of parallelism within the default ALPS timeout of 60s.

The workaround is to set $PMI\_MMAP\_SYNC\_WAIT\_TIME = 300$ in the application environment, which will increase the timeout to 300s instead of 60s. However, the solution was a feature that SchedMD implemented in later verions of 15.08 which merged the functionality of srun and sbcast (srun –bcast) to automatically copy the executable prior to execution. In 16.05 a further improvement of this to enable compression is coming. That is expected to put srun performance on the same level as aprun job startup

performance (in combination with CLE6.0).

4) Expected deployment changes upcoming for CLE6.0: In CLE6.0 the shared root will not exist. Instead things are moving to a site-customizeable prescriptive image-based scheme. The "netroot" image capability will allow sites to determine which files are memory-resident and which are network accessible. The root filesystem of the node is writable, and thus updateable.

Our current plan, which we will begin to test in the coming weeks, is to migrate our deployment machinery to supporting RPM installation, but with a "–prefix" of /usr. This should simplify the deployment as well as the version management. Updated versions of the SLURM RPMs can then be performed between jobs.

Since we can specify that SLURM should be memory-resident, we are expecting an improvement in performance since SLURM depends strongly of dlopen()ing shared objects within the SLURM lib directory. Based on some initial timings, this may reduce each srun on large scale systems by as much as 4 seconds.

Deployment of the SLURM configuration files is less well understood. Those may remain in a network-share to expedite updates of those more-frequently-updated files.

C. Interactions with Cray Node Health Check

The Cray Node Health Check is run by SLURM and it does perform all of its existing functionality. However, it is run from the slurmctld node following the completion of all slurmds in the allocation. SLURM is quite aggressive about cleaning the node and terminiating all processes prior to allowing the NHC to run. This has greatly reduced the quantity of issues the NHC discovers. However, the concept that the NHC must be run from a central location across the entire allocation (ALPS reservation) simultaneously is somewhat conceptually at odds with SLURM. SLURM has features which can allow a job to reduce (or increase) the size of the allocation dynamically. SLURM can return nodes to the scheduling pool while some are still trying to clean up. These features could allow us to improve utilization and user productivity greatly. With the current way the NHC runs, all nodes must complete prior to running the NHC, and cannot move to new jobs until the NHC runs. This means that if one process on one node is stuck in an unkillable state (waiting upon disk IO, for example), that all of the remaining nodes in the allocation will be held idle until that clears up or is manually handled by an administrator (e.g., by marking the node "down" in SLURM). We believe that if the NHC could be run separately for each node, perhaps in the epilog of the job, it would continue to deliver value, but would be more compatible with the SLURMish way of doing things. We hope to work with Cray and SchedMD to make this workflow a reality in (near) future versions.

III. SCHEDULING

We have spent significant time and resources learning how to most effectively schedule the NERSC workload using SLURM. We have a great many different codes that are run on the systems, and NERSC users take advantage of every scale. We typically have on the order of 100s (edison) to 1,000s (cori phase 1) jobs running, with a queue at least 10x larger than the job count. This means that we have a large dynamic range of jobs to schedule, with different job-class-based priorities in the mix. The edison system runs a large quantity of small jobs, but the focus is on large jobs consuming at least 1/8th of the system (683 compute nodes). Cori phase 1, however, is our "data" partition which is is more focused on a High-Throughput style workload, which a large quantity of "shared" and "realtime" jobs.

A. Backfill and Utilization

SLURM has two scheduling algorithms it uses. Because of the large dynamic range in the scale of the NERSC workload, almost all jobs are started by the "backfill" scheduler. The backfill scheduling algorithm is, by default, very conservative, but does respond very effectively to changing system conditions.

It has no memory from one cycle to the next and always works down the priority-ordered queue of jobs. The algorithm works by first sorting the queue of jobs, then determining the earliest time the highest priority job can run. That start time is then a constraint for all future scheduling operations in the backfill run (for all intents and purposes, could be considered a "running" job, just in the future). The next job is then considered with all the current running jobs and the planned future job and so on. Thus the SLURM backfill logic will start any job so long as it does not delay any higher priority job.

Furthermore, only the backfill scheduler can start jobs out of order. The combination of these behaviors means that SLURM backfill is effectively infinite resource reservation depth scheduling – which is very computationally intensive.

To prevent the scheduler from ineffectively draining the system to start jobs, it is therefore extremely important that the priority function is stable if there is any large range of job sizes being considered. This is because the calculation of the schedule is path dependent, meaning the sort of the jobs should stay the same from backfill cycle to the next backfill cycle.

While this method is very effective in responding to changing conditions on the system, it also means that the high priority portion of the job queue has probably been examined many times. Meaning that from a backfill perspective, the high priority segment of the queue is "low value" – if a job couldn't start on the last cycle it is unlikely to be able to start on this cycle. It is the low priority segment of the queue that presents the most recently submitted jobs

(in our priority scheme) and thus represent the most likely candidates for backfill.

Thus, working with SchedMD, we designed new functionality implemented in NERSC deployments of SLURM, and available generally in 16.05, which add the "bf_min_prio_reserve" backfill parameter. This adds a specific priority level above which the scheduler will reserve resources (i.e., add scheduling constraints), and below which jobs will simply be tested to see if they can start "now". This effective relaxing of scheduling policies enables small jobs to start very quickly maximize the backfill opportunities realized on the system. We observed an average increase of about 7% utilization by using this algorithm for our workload.

In the future, we plan to work on methods to manage priority levels to better manage how jobs proceed from the low priority to the high priority segment of the list.

IV. NERSC Customizations to SLURM

SLURM is itself fantastically customizable and highly configurable, and NERSC has tried to take full advantage of that to introduce desirable functionality. One particularly useful tool in the SLURM software architecture is that of the SPANK plugin framework. This enables site-configurable plugins to interact with the batch system, prior job submission, during job allocation, and following job conclusion.

A. Custom SPANK Plugins

We have implemented two SPANK plugins. The first is actually a component of "shifter" [2] and enables tight integration between SLURM and shifter. That is generally available as part of the shifter open source project.

The second plugin bundles two important capabilities:

- enable users to run Intel vtune amplifier within their job
- perform client-side quota-check validation prior to allowing job submission, used to gate job submission if the user has exceed a soft quota

These functionalities are best introduced as SPANK plugins because they require tight integration into the job submission and allocation logic, as well as ensuring that the work is distributed across the resource, rather than concentrated on the single slurmctld node.

If a user requests the vtune capability at job submission time (using the custom "–vtune" flag to sbatch), a vtune-specific prolog is run during job allocation to load the needed kernel modules. At the conclusion of the job the kernel module is unloaded. It is important to dynamically load and unload the vtune kernel module, and to only allow node-exclusive jobs to access this functionality both from a security and system stability perspective.

Lustre quota checks are performed entirely within the SPANK callouts when sbatch, salloc, or srun (not as part of an existing job allocation) are executed by the user. Were lustre MDS slow or nonresponsive, it is considered better to distribute that work rather than centralize it as part of the direct job_submit.lua filter.

B. Accounting

NERSC makes heavy use of the slurmdbd accounting system, in particular the job execution limit functionality that it can enable. However, NERSC has long maintained an in-house account management database, NERSC Information Management (NIM). NIM implements all of the NERSC business logic and coordinates the many project allocations and users across the center. We have written a set of python scripts that synchronize the NIM and SLURM databases, supporting both in-bulk replication, and replication on a fine-grained basis in coordination with the job_submit.lua script.

These scripts create the following in the SLURM accounting database: accounts, users, associations between accounts and users, QoS access lists, per-association TRES limits (only for BurstBuffer). A secondary redis cache is updated to track account and association balances. The redis cache is used to perform out-of-band enforcement of limits because we do not reject jobs for out-of-time users, but rather route those jobs into a very low priority QoS ("scavenger") through the job_submit.lua script.

The NIM accounting system runs a detailed sacct query nightly to accumulate the daily usage for billing. NIM also directly queries the SLURM mysql database to charge for reservation time.

C. Job Submit and Update Plugin

SLURM provides a site-configurable job_submit plugin, which allows a site to review and potentially modify jobs prior to acceptance into the queue. NERSC uses the lua[3]-based job_submit plugin supported by SLURM to allow fairly flexible and updatable (changing a script is much easier than rebuilding SLURM) review and modification of jobs.

Our job_submit.lua is primarily engaged in validating that the user is allowed to access the functionality they have requested. This includes verifying that the user is known in the SLURM database and, if not, dynamically attempting the NIM $\leftrightarrow$ SLURM integration to lookup that user. It estimates the quantity of resources requested by the job and checks against the redis cache if the user has sufficient balance to access those resources. If not, the job may be rejected or modified to go into the low-priority "scavenger" QoS. The job_submit.lua script is also used to implement and validate our network filesystem "licenses". When a network filesystem is unavailable, either due to maintenance or misfortune, an administrator can set the number of these licenses available to zero, preventing jobs that would require the file system from starting. If the job requests the shared partition, job_submit.lua will remove the craynetwork GRES

to disable access to the aries network, thus enabling more than four concurrent jobs per node.

D. User Accessibility

1) sqs: "sqs" is a NERSC custom batch queue monitoring script that provides basic batch job info plus the job ranking information. The original version provides priority ranking based on start time estimated by the backfill scheduler. Most of the jobs in the queue do not have a start time hence no ranking info. In order to provide users with more information of their jobs position in the queue, a new version of sqs was deployed with two columns of ranking values to give users more perspective of their jobs in queue. Job priority ranking with absolute priority value (a function of partition, QOS, job wait time, and fair share is also provided.

2) MyNERSC: The NERSC website integrates large amounts of information from SLURM via the MyNERSC [5] portal. This information is collected and integrated with a decades worth of batch information from previous NERSC systems to give users a great deal of current and past information. The aggregation of this data is done by merging information collected from sacct, direct database query, sqs, scontrol, and other job completion sources. The data is merged into the NERSC jobcompletion database. That database is then used to deliver information, via MyNERSC, about current queued jobs, past progress of those jobs in terms of prioritization, starting time estimates, and finally information about past job resource usage and performance. This data is also integrated into a dashboard showing other NERSC resources such as parallel filesystem utilization and performance monitoring. Users can additionally submit, hold, release and delete jobs from the MyNERSC web-portal - this functionality utilizes the NERSC API (NEWT) [6] to communicate with the SLRUM scheduler via a REST API.

V. Transitioning to SLURM: User Support Perspective

Overall our SLURM adoption for users has been relatively smooth. In order to help users, we have provided detailed documentation on SLURM in a transition guide, containing example batch scripts and tutorials. NERSC staff has worked directly with some specific applications and users to ease porting particularly complex workflows.

Many features of SLURM gives users more flexibility in running their applications, for example: No separate partitions for "premium" or "low" priorities etc. These are now available via QOS settings in the "regular" partition. There is no need to support CCM (Cluster Compatibility Mode) [4] applications in a separate partition. It is now supported via a custom SLURM plug-in, leveraging Shifter to provide this capability.

One particularly noticable change for users with the SLURM transitioning is that hyperthreading is on by default, so SLURM sees all the logical cores. Submitting a job with "#SBATCH -n" but without "#SBATCH -N" will get half the number of nodes desired. For a hybrid MPI/OpenMP program (if compiled with OpenMP enabled), when running in pure MPI mode, without setting OMP_NUM_THREADS to 1 (for Intel and GNU compilers), the program will run with 2 threads implicitly.

Multiple iterations of task/CPU-binding configurations were tested during our early user period on Cori. With our current setting (slurm.conf TaskPlugin=cgroup,cray; SelectTypeParameters=CR_SOCKET_MEMORY,OTHER_CONS_RES), when an application specifies number of nodes (-N), number of MPI tasks (-n), and number of OpenMP threads (set via OMP_NUM_THREADS environment variable), the default process and thread affinity with srun is good. Users need to explore with advanced settings for more complicated binding options, such as memory binding, different MPI tasks and threads per node (MPMD), etc.

One issue is that SLURM has no job "idle limits" can be used to limit the quantity of jobs per queue eligible for accruing priority. In SLURM, all jobs in the queue are eligible for priority accrual, except held jobs and dependent jobs. To ensure fair access to system resources, other queue policies such as submit limits and the great variety of SLURM run limits must be used.

VI. Conclusion and Future Directions

NERSC has consistently delivered highly usable and highly utilized systems with SLURM. It has enabled new functionality on our existing hardware, and its open-source nature allows a close collaboration with the vendor (SchedMD) to work through issues rapidly. Typically, our incident reports have been resolved within a day, frequently with a modification to the SLURM code-base to better meet our needs. The introduction of a site-controllable agent on the node, in the form of slurmd, and the embrace of more Linux-standard environments has greatly enhanced the manageability of the system and its usability by users.

Looking forward, our immediate challenge will be to deploy SLURM 16.05 in CLE6. There are some significant expected changes in our deployment strategy. We anticipate no loss of function and hope for a substantial gain in performance. Following that, we will embark on integrating Cori Phase 2 into the existing system, introducing 9,300 Intel Knights Landing nodes. This will present several new challenges, most especially that of managing a heterogeneous system with a heterogeneous workload (HPC + Data). The KNL makes new scalability demands on the slurmstepd relative to the Haswell in terms of thread management. Special attention and management will be required to support user-requested hardware modes, possibly requiring compute node reboots to shift between modes. The scheduler will have to manage the timing of these reboots to intermix in the queue most effectively. Cori phase 2 will be the largest quantity

of schedule-able units within a single system that NERSC has ever deployed, and we anticipate new and interesting challenges in scaling to this level.

ACKNOWLEDGMENT

The authors would like to thank all the folks at SchedMD, in particular Danny Auble, Brian Christiansen, Moe Jette, and Tim Wickberg for their rapid and effective support throughout the NERSC SLURMification.

The authors would also like to thank Terence Brewer, Brian Gilmer, Mark Green, Robert Johnson, Shawn Le, Steve Luzmoor, Pete Martinez, and Randy Palmer from Cray for their advice and continuing support.

The authors thank and are greatly appreciative of the contributions of everyone at NERSC, but in particular those of Katie Antypas, Brian Austin, Deborah Bard, Wahid Bhimji, Tina Declerck, Lisa Gerhardt, Scott French, Richard Gerber, Rebecca Hartman-Baker, Steve Leak, Ian Nascimento, Dr. R. K. Owen, Zhengji Zhao to making SLURM a success on our systems.

The authors would finally like to thank all the NERSC users for their many helpful bug reports and patience during the transition to SLURM.

REFERENCES

[1] SLURM, http://slurm.schedmd.com

[2] Canon and Jacobsen, Contain This! Unleashing Docker for HPC. Cray User Group, 2015

[3] LUA Programming Language 5.1, https://www.lua.org/manual/5.1/

[4] Using Cluster Compatibility Mode in CLE, http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/chapter-9b6qil6d-craigf.html, Cray Inc.

[5] MyNERSC - NERSC User Information Portal, http://my.nersc.gov

[6] NEWT - NERSC Web Toolkit, http://newt.nersc.gov