Deploying SLURM on XT, XE, and Future Cray Systems

Gerrit Renker, Neil Stringfellow, Kris Howard, Sadaf Alam and Stephen Trofinoff
Swiss National Supercomputing Center (CSCS)

ABSTRACT: *We describe porting the open-source SLURM resource manager to the Cray BASIL/ALPS interface; and report on experiences of using it on our main 20-cabinet Cray XT5 production platform, as well as several development systems of a heterogeneous multi-cluster environment.*

KEYWORDS: resource management, scheduling, SLURM, ALPS, XT5, XE6, GPU

1. Introduction and Motivation

The Swiss National Supercomputing Centre (CSCS) operational facility hosts a number of Cray MPP platforms and has traditionally been an early adopter of Cray technologies, including the first XT3 installation in Europe, and more recently the XE6 and XMT2 platforms. In order to guarantee optimal availability, utilisation of supercomputing resources, and to ensure that stakeholder criteria are met, CSCS has developed a highly customized resource management, scheduling and accounting environment. Until recently, this was based on Altair's PBSPro [PBSPro2011], which allowed replacing the internal default scheduler with a site-specific variant written in Tcl [Welch2003]. The initial skeleton of the Tcl scheduler used at CSCS was developed by Jason Coverston of Cray, and over the years this has grown into a sophisticated tool hosting in-house algorithms to enforce the centre's evolving policies. However, recently Altair announced significant modifications to the scheduling interface in future versions of PBSPro whereby the option to use Tcl as a scripting language will no longer be supported. This leaves scriptable Python hooks, called at certain points by the system, as only possibility to customize scheduling policies. Extensibility and flexibility are thus reduced by vendor-provided abstractions.

In order to maintain the flexibility of adapting the system as the centre's policies continue to evolve, CSCS started to look for alternatives that, in a similar manner, would support:

1. Aggressive backfilling.
2. Bottom-feeder policies (users can still run in a very low priority mode, even after exhausting their quota).
3. Extensions to support a key customer (MeteoSwiss), tailored to the demanding requirements of operational weather forecasting, delivered several times per day and on-demand.
4. Robustness extensions (scheduler health monitor).

Having to move forward without the use of our customized Tcl scheduler, CSCS considered a wide range of batch and resource management offerings as viable candidates. These included batch systems currently in operation at other Cray sites, such as Altair's PBSPro, Platform LSF, Torque with Moab, but also alternatives not yet deployed on Cray XT/XE machines.

Our investigations into scalable resource management software for use on Cray systems identified SLURM [Jette2003] as the most promising candidate, since it fulfils the key requirements namely *extensibility*, *flexibility*, and *maintainability*. From its initial design phase SLURM has been written with *scalability* as a central criterion, and has been deployed on many of the world's largest supercomputing systems. These include the top system in the world according to [Top500-2010] (a heterogeneous, GPU-based system), and Europe's largest supercomputer, at CEA in France. Moreover, the developers are targeting even larger systems, including the 20 Petaflop/s BlueGene/Q system to be deployed in 2012 at Lawrence Livermore National Laboratory.

The peer-reviewed open-source code is available for a range of platforms and favoured by a substantial development community, whose activity on a large number of systems is evident in the SLURM mailing list archives.

Since early 2010, CSCS has been porting and testing SLURM on various XT, XE, and non-Cray cluster systems; with cluster dimensions ranging from one or two nodes with a few cores up to a 20-cabinet Cray XT5. Patches evolving from this development have been reviewed and accepted by the main SLURM developers (of SchedMD, LLC), who continue to evolve the interface and actively provide ongoing support for the Cray port of this modern, multi-threaded resource manager and scheduling system.

Figure 1 shows the *general SLURM architecture*. On a regular cluster, there is one slurmd execution daemon per compute node, centrally controlled by a slurmctld with failover possibility.

Users have a list of *commands* and a graphical interface called *sview* available to submit and query the system. Accounting is built-in via an integrated MySQL/Postgres interface, where job run information can be tracked directly.

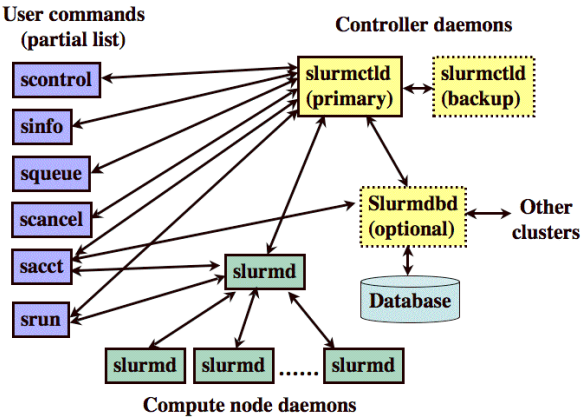

Figure 1 - SLURM conceptual architecture
(source: courtesy of SchedMD, LLC)

SLURM provides two optimized and priority-directed mainstream *scheduling algorithms*, FIFO scheduling and conservative backfilling, out of the box. Like many other parts of the highly configurable architecture, the scheduling interface is abstracted into a SLURM plugin (other scheduling plugins include wiki and wiki2, used by Maui and Moab respectively).

Although hooks are provided to communicate with other schedulers, the standalone scheduling performance of SLURM has so far proven to be more than enough for the needs of our centre. In particular, the recent addition of a database-driven *multi-cluster feature* allows to cross-submit jobs from one cluster to the other (useful for our post-processing jobs), and from remote bastion hosts.

The *SLURM database* can be used both passively (classical accounting) and actively, in order to enforce various limits, quality-of-service, and fairshare settings. The latter further enabled us to customize the way the scheduling

priority is derived, using the powerful abstractions of the flexible *multifactor-priority plugin*. This was an added bonus, since in fact it took very little effort to implement the centre's customized allocation policy.

The quality of the source code sets an example by not only being state-of-the art, but also at an extremely high standard throughout. It is future-proof by providing *scriptable library interfaces* in C, Python, Perl, and Lua. Furthermore, there are *plugin hooks* into nearly every interesting aspect of the resource manager architecture (job submission, job priority, scheduling, node selection, network topology, interconnect switch type, MPI type, generic consumable resources, checkpoint/restart type, task execution, process tracking, accounting, plus additional plugin architecture for job and node control (SPANK)).

Considering all these advantages, CSCS funded porting efforts to run SLURM on XE (since June 2010) and several XT systems. Cray XT/XE systems employ a uniform batch system interface called BASIL/ALPS (see next section). Porting consisted in mapping the SLURM abstractions onto the 4 methods provided by BASIL; and in figuring out the undocumented (proprietary) procedural details of ALPS operation required for smooth and robust interaction with a scheduling system.

This report summarizes our steps leading up to the successful deployment of SLURM on CSCS day-to-day production machines, including a 20-cabinet XT5 production platform and a 2-cabinet XE6 research and development system.

The document layout is as follows: section 2 provides porting and implementation details. We report on initial centre-wide deployment experiences with SLURM on Cray/non-Cray clusters in section 3. Extensibility for future platforms is discussed in section 4. Section 5 then concludes the paper.

2. Porting SLURM to XT and XE

Placement of executable MPP codes onto Cray XT/XE compute nodes is handled by the Cray *Application Level Placement Scheduler* (ALPS) whose interface to third-party batch scheduling systems is specified by the Cray *Batch and Application Scheduler Interface Layer* (BASIL). This is a static XML interface consisting of 4 XML-RPC calls: QUERY (get current placement inventory); RESERVE (request nodes); CONFIRM (acknowledge RESERVE request); RELEASE (indicate to ALPS that the job should now be finished).

The operational details of this interface are proprietary, only some aspects have been published [Karo2006]. In this regard we are indebted to the exceptionally helpful and informative support that we have experienced from Cray when faced with operational questions that arose in practical experiences with ALPS. Otherwise, the absence of documentation for behavioural details would have been more of a "trial and error programming". Merely implementing the XML calls does not produce an operational interface, let alone provide fault tolerance.

The fact that the entire batch layer is defined by a single XML-RPC interface means that all ALPS-based schedulers are essentially isomorphic. The situation is comparable to the mandatory basic health insurance in Switzerland: since every health insurance company is bound by the same basic requirements, the insurance companies in effect all offer the same package, at various prices.

Under the hood of *Cray ALPS/BASIL* is a

distributed client/server architecture consisting of multiple daemons that keep common state through a MySQL database and memory-mapped, NFS-shared files.

We are not concerned with the details of ALPS daemons, since these are described by Cray documentation, and have also been published elsewhere [Karo2006]. The two main entry points of interest are: *apbasil*, which takes the XML-RPC calls on its stdin and writes a response to stdout; and *aprun*, which remotely launches applications once an ALPS reservation has been set up through the BASIL CONFIRM call. Since ALPS releases a reservation only when there are no outstanding claims (running applications) against it, recent additions to increase robustness also employ *apkill* in order to clean up orphaned ALPS reservations after the terminal (COMPLETING) stage of a job.

Figure 2 shows the interplay of various ALPS components illustrated on the PBS architecture. As per the above comments, SLURM mode of operation is essentially identical; just substitute *sbatch* for qsub, *slurmd* for pbs_mom, and *slurmctld* for PBS server/scheduler.

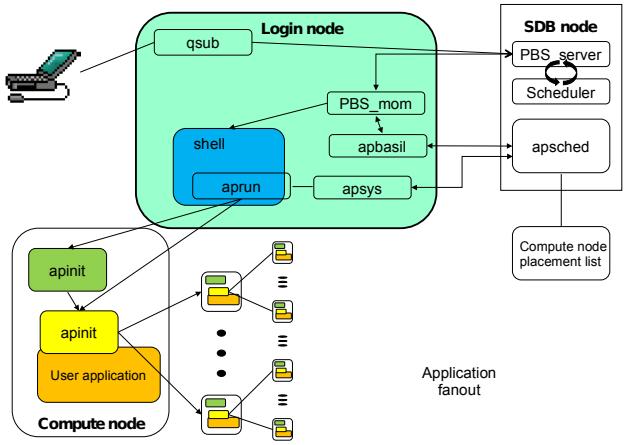

Figure 2 - ALPS interaction with PBSPro
(source: courtesy of Rick Slick, Cray Inc.)

Our port was *non-intrusive* in not modifying the generic architecture and command set of SLURM for the sake of a specific platform.

The single exception to this rule is that, since *application launch* is under the control of ALPS, the SLURM *srun* command can not be used for dispatch, control, and accounting of job steps. At CSCS this currently means using *aprun* in its place. Thanks to development effort at ORNL, there is now also an srun-like wrapper around aprun which implements the behaviour of *srun* to the extent possible. The wrapper is available in contribs/cray/srun.pl and also in a separate rpm.

Keeping the port self-contained and non-intrusive is thanks to the modular *select/cray plugin* provided by SLURM developer Danny Auble. Architecturally it is deployed as a *node selection plugin*, whose clever piggybacked design allows to also accommodate all the remaining Cray-specific abstractions.

Typical node selection functionality comes into play before a job run (setting up and confirming the ALPS reservation), and afterwards (returning the job nodes via the BASIL RELEASE call).

The actual *SLURM-specific node selection operation* is deferred internally from the select/cray plugin to the attached select/linear plugin. This is a tried and tested variant that always allocates entire nodes to jobs. As of now, no other allocation mode (such as for instance select/cons_res based on consumable resources) can be served, since ALPS from the beginning has always only supported allocating entire nodes to jobs. A shared allocation or over-subscription of resources, such as provided by the select/cons_res plugin, or task affinity options, are likewise not possible with the

current state of ALPS. For these reasons, the select/linear interface employed internally by the select/cray plugin provides more than enough functionality to match the required data input of ALPS.

With regard to *interactive mode*, there is a major difference to PBS-based systems (figure 2): in those systems, batch and interactive jobs use the same execution environment and launch command (qsub), while SLURM offers dedicated commands (sbatch and salloc) that stand for two fundamentally different modes of operation [Jette2003].

In *batch mode*, job scripts execute as usual on a remote service node running slurmd, whereas *salloc* spawns the interactive session directly on the current login node. It is thus giving the user access to exactly the same environment as in a normal login session. Interactive jobs on PBS-based systems, in contrast, perform session forwarding to a remote execution host. The intuitive appeal of salloc lies in providing full MPP facilities within the customary environment of the user. This proved very beneficial for beginning users, debugging, and interactive refinement of job scripts.

Since on Cray systems it is not possible to directly launch applications on compute nodes, SLURM supports the traditional architecture of dedicating specific *"MoM" service nodes* to execute the job scripts, and then use the ALPS infrastructure to defer actual application launch onto remote compute nodes. In PBS-based systems the job launcher daemon is called *pbs_mom*, which in SLURM corresponds to *slurmd* in frontend mode. Thanks to recent extensions by SLURM developer Morris Jette, a fault-tolerant architecture of multiple such frontend nodes is now part of SLURM (the earlier single-frontend mode might have caused load concerns on older XT service nodes).
An interesting detail of the of the select/cray plugin is in also providing the abstractions for *network topology* and *type of interconnect*. Such functionality is normally provided by the separate *TopologyPlugin* of SLURM, which deals with the topological details of the interconnect. In Cray terms this means torus dimensions (2D/3D), class of rack/cabinet cabling, and type/version of the interconnect (SeaStar, Gemini, or Baker). Though this design choice meant some overloading of the node selection plugin, it has helped to keep all Cray-specific abstractions confined to a single place (rather than adding a separate topology/cray plugin). Its operation is transparent to the user: the same plugin is used for XT and XE architectures, with differing cabling classes and torus dimensions.

Accordingly the following *topology operations* are performed by the select/cray plugin. During initialisation, it picks up the currently configured ALPS_NIDORDER, so that nodes are selected in exactly the same order as ALPS would. As a consequence, recent Cray work on enhanced ALPS node placement [Albing2010] can directly be leveraged within SLURM.

A second topology operation of the plugin is to *resolve the (X,Y,Z) coordinates* of each compute node (on 2D systems the X component is always zero). These node coordinates are visible as the virtual *NodeAddr* attribute of SLURM nodes, which can also be viewed in graphical SLURM tools such as *smap* or *sview*.

After 8 months of deploying and refining the use of SLURM on several smaller Cray clusters at CSCS, we migrated our main 20-cabinet XT5 production system in April 2011. Despite the usual skepticism that always accompanies change, our experience of rolling out SLURM

has in fact been smooth, efficient and trouble-free. The excellent code quality and its inherent scalability meant that we were basically able to migrate our 2-cabinet test setup onto the 10 times larger production system with very little change.

During initial roll-out we experienced one of several "*wrong guesses*" as to the operational behaviour of ALPS. On our larger XT5 with CLE 2 the use of *apkill* to clean up orphaned ALPS reservations proved fatal unless first the reservation is cancelled (a behaviour that had not been observed with CLE 3 on the XE).
We then were in a position where we had to deploy a *fix on a live system* on which jobs continued to run. The SLURM developers have anticipated such an eventuality and thus provide capabilities to also deploy rolling upgrades on live systems. Owing to the high quality of the code (actually using a developmental pre-release), we were able to halt SLURM in mid-operation, unpack the tarball with our fix, and bring SLURM back up again, without losing even a single of the several hundred running jobs.

CSCS is currently using SLURM on 6 *different clusters*, featuring XT, XE, x86, and GPU-based systems. Many of the clusters are subject to *centre-wide policies* such as allocation quotas, admission control, and usage monitoring.

Despite a host of very specific requirements (e.g. prime-time/non-prime time mode, allowing coexistence of research and production use of one and the same system), the inherent flexibility of SLURM allowed us to implement our many detailed requirements with very little administrative effort, and appreciably little downtime. The usual pain of mapping the abstractions of one vendor to another is noticeably absent: SLURM proved very system administrator friendly, and provides usually more than one way to accomplish the same task.

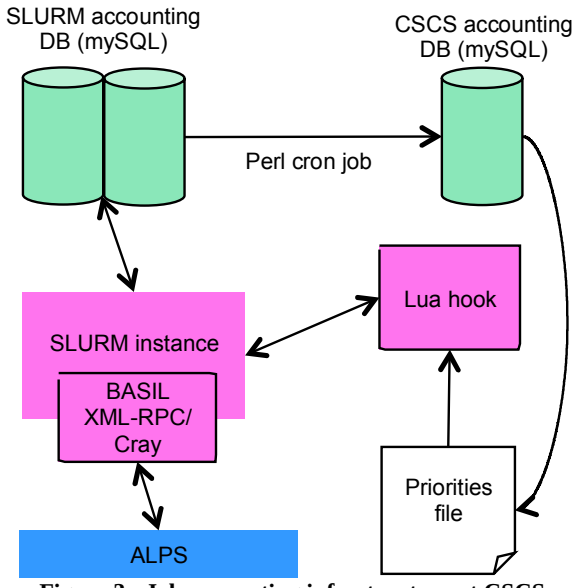

Figure 3 - Job accounting infrastructure at CSCS

Figure 3 shows a bird's eye view of our centre-wide SLURM deployment. A central database contains user and project information, controlling also the per-group allocations.

No more than 180 lines of Perl were required to extract the requisite accounting information from the job data tracked by SLURM (accounting parsers for batch vendor logfiles, in contrast, usually took several months of development).

CSCS uses 3-month *project quotas* and offers its users a *bottom-feeder policy*: projects which have exhausted their quota may continue to run jobs, but only at an extremely low priority, below any other legitimate user.

This requirement necessitated a dynamic modification of SLURM's *scheduling priorities*, accomplished by just a few lines of lua code, thanks to the job_submit/lua plugin included with SLURM. From an external text file, read at job submission time, we set the base 'nice' value

for the scheduling priority. Additional factors, such as ageing and preferring larger jobs over smaller ones, are taken care of by the priority/multifactor SLURM plugin, tunable at runtime via configurable weights.

A few additional lines of job_submit/lua code ensure that disabled projects are banned from running jobs, and that the right prime-time/non-prime-time partition is selected. Given that even complex administration tasks were handled with such little effort, we anticipate further required custom modifications with confidence and ease.

Being a customer site, we were not privy to information usually available to batch vendors. *In second-guessing undocumented operational details of the ALPS interface we repeatedly erred in our guesses.* For example, the number of nodes reserved by ALPS not only depends on the number of processing elements (-n), thread depth (-d) and number of processing elements per node (-N), but also on the per-PE memory (-m). Hence it happened, after continuously running our in-house scheduler for over 1 year, that an error became evident in correctly deriving the required number of nodes also from the per-PE memory. As in all other questions, we are very grateful to the Cray team for their efficient and informative help on clarifying what had gone wrong in our initial idea of how ALPS interprets allocation parameters.

3. Experiences deploying SLURM on Cray and non-Cray clusters

As a national HPC centre we have been looking at the migration to SLURM from more than a single point of view. In this section we detail one of the main requirements in that regard: how an end user can interact with a complex programming environment via the SLURM interface vs. the PBSPro interface (which previously had been used on the majority of CSCS systems). We also contrast how resources can be managed and controlled on a commodity multi-core cluster, where the ALPS middleware is not present.

The Cray XT5 system is composed of dual-socket six-core Istanbul nodes with SeaStarII interconnect. A user can control *mapping of MPI tasks and OpenMP threads* onto sockets and nodes (depending on memory and optimization requirements) by means of *batch script parameters*, which enable ALPS to make an appropriate reservation, and *arguments to the aprun command* that describe the claim of an individual application on compute nodes.

PBS Pro example:

```
#!/bin/bash
#PBS -l mppwidth=1
#PBS -l mppnppn=1
#PBS -l mppdepth=8
#PBS -l walltime=00:30:00
#PBS -V
cd $PBS_O_WORKDIR
aprun -n 1 -N 1 -d 4 ./exe
```

SLURM example:

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --time=00:30:00

aprun -n 1 -N 1 -d 4 ./exe
```

A more complete comparison of PBS and SLURM directives is available at http://user.cscs.ch/running_batch_jobs/rosa_cray_xt5.

In terms of monitoring and querying their jobs, users have familiar interfaces available. The output from *squeue* command (corresponding to

qstat in PBS) illustrates this:

```
JOBID USER      NAME       ST START_TIME  NODES
12214 blofeld   evilplot   PD   17:51:55    176
12215 blofeld   evilplot   PD   18:51:55    176
12216 blofeld   evilplot   PD   19:51:55    176
12217 blofeld   evilplot   PD   20:51:55    176
12246 blofeld   evilplot   PD   23:21:55    176
```

Furthermore, SLURM's *sview* tool provides different graphical interfaces; for instance *job view* (figure 4), or *node view* (figure 5). The left side of sview's display shows the specific nodes associated with a job or other entity in a two- or three-dimension format, revealing its topology.

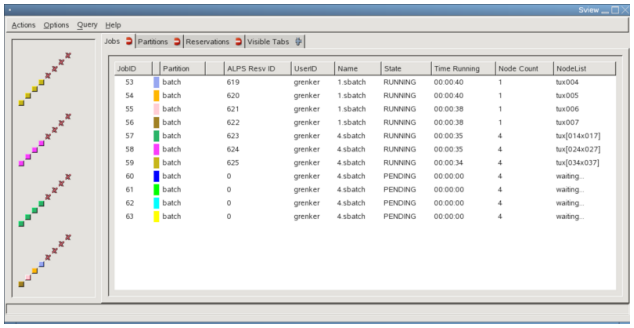

Figure 4 - `sview` job display on a single-chassis Cray XT5

sview provides complete user control over the information displayed, including the fields shown, columns used and sort order. It is also capable of displaying information about all clusters on a site.

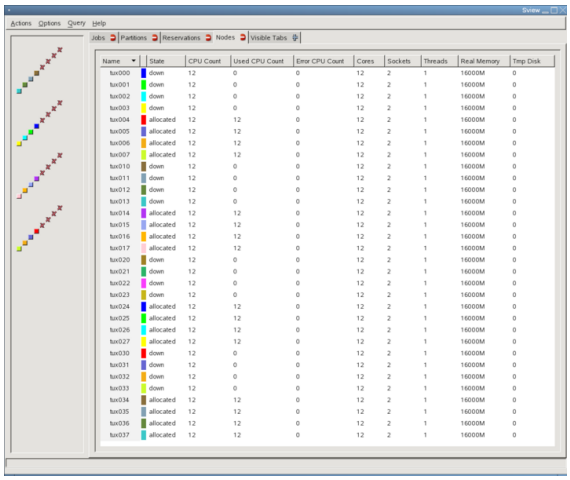

Figure 5 - `sview` display of node usage on a small Cray XT5

In addition to porting SLURM to the Cray XT and XE architectures, CSCS has begun a site-wide *deployment of SLURM onto a number of other operational clusters*, from standard multi-core/multi-socket configurations to clusters whose nodes incorporate GPUs. In general, the configuration and operation of SLURM on these clusters has shown itself to be straightforward. As these clusters are frequently used for small jobs that only require some fractional part of the resources available on a node, it is desirable to *allow multiple users to share nodes* under the direction of the resource management infrastructure.

CSCS is currently investigating the potential of SLURM's various *affinity plugins* on these clusters. On a Cray system the vendor has full control over the hardware and software stack in order to provide optimal facilities for task mapping, including a specific core-specialization mode.

Owing to the generality of its design, SLURM can just as well be deployed on commodity clusters, where, in addition to the flexibility of OpenMP runtime environments, there are choices for *MPI implementations*, such as OpenMPI, MPICH2 and MVAPICH2. These open-source MPI implementations have *processor mapping and affinity options available by default*, which work transparently with process managers and launchers such as srun, mpirun, mpiexec, or mpiexec.hydra.

We are continuing our investigation of SLURM features on complex clusters with heterogeneous sets of nodes, including GPGPU devices.

4. Discussion: Extensibility for Future Systems and Programming Environments

Using SLURM on Cray with ALPS and on multi-core/multi-socket, as well as GPU clusters has made us ask what the future ALPS interface might look like. Already the abilities are quite restrictive: one executable per node and no shared allocation mode. In addition, it is not clear how (multiple) GPUs are to be used with ALPS.

A key experience has been that the shape of the ALPS interface has prevented us from taking advantage of several powerful SLURM features that would be extremely valuable for application developers and production science users alike: such as running multiple executables on a node, oversubscribing resources for certain types of scalability tests, and many recent concepts, such as the upcoming use of cgroups [cgroups2011].

On the Cray systems we have had to make compromises on what can be offered on a commodity cluster deploying SLURM, by reducing batch system interaction to the lowest common denominator of the select/linear interface. We may not use select/cons_res to perform node selection on Cray, and likely are constrained in developing future extensions, such as node selection based on GPU features (generic consumable resources)[1].

A major advantage of the native SLURM application launcher is the *ability to share nodes*, either by having multiple executables running simultaneously, or by placing multiple different jobs on a node. Notwithstanding the primary purpose of Cray's XT and XE systems for massively parallel applications, it is sometimes expedient to run serial or small-scale parallel applications on such a system, whether for post-processing or data analysis, for auto-tuning, or because the installation has to cater to a wider community of users in the place of a traditional cluster. However, the ALPS job and application launchers require that each executable be placed on a separate node. This leads to a *waste of resources*, a situation which also arises when using a MPMD style of execution.

Consider the innovative field of *code auto-tuning* as an example. The complexity of modern node architectures makes it difficult or impossible for either an application developer or a compiler writer to determine the *best set of optimizations* for individual code kernels, and so a large number of developers are turning to auto-tuning to derive the best executable for a given problem. Auto-tuning for a serial kernel necessitates that *a very large number of similar code fragments are run* in order to find the best variant. Since however the current ALPS infrastructure does not support running more than 1 executable on the same node, the developer typically fills the node with multiple copies of the same kernel. Whilst on a dual-core machine this might have been considered an acceptable loss of machine efficiency, in an era of 24 and 32 core nodes the accumulated waste of resources renders such a system economically unattractive for code auto-tuning.

SLURM's application launcher further allows the *oversubscription of node resources,* so that jobs may share cores (for example, if the job's CPU requirements are rather weak, or if applications are more memory-bound than CPU-bound). Although such is not a typical use-case for a Cray MPP system, the ability to oversubscribe resources is a great asset in *testing parallel applications with larger numbers of ranks than the number of cores*

[1] https://computing.llnl.gov/linux/slurm/gres.html

available on a machine. This has previously proven to be useful when Cray's own MPI developers were running at 150,000 ranks on a 30,000 core machine [Pagel2009], using a special 'emulation' mode of ALPS. In contrast to SLURM, this method of operation has not been made available to the interested user community.

In terms of large-scale resource management and information gathering, the *ability to collect accounting information at both the level of jobs and applications*, a default provided by SLURM, is invaluable to compute centre managers and system administrators alike. A resource management system that provides full accounting transparency allows decision making to be taken based on the *amount of resources consumed for an individual project* through job accounting; in addition it exposes *how individual applications are launched and the resources attributed to them*, by collecting individual application run statistics. With the current ALPS infrastructure it is necessary to trawl through a number of system log files in order to collect the relevant statistics; a cumbersome task such as has been carried out also at other centres [Maxwell2008], [Fahey2010].

5. Summary and Future Plans

We demonstrated and discussed the deployment of the SLURM infrastructure on Cray XT/XE systems, and use of accounting data within the existing environment at CSCS.

We have identified a number of important pieces that are currently missing in ALPS, when compared to what is available in SLURM; their inclusion would be invaluable to either application developers or compute centre staff.

These omissions include allowing *multiple jobs or applications per node* and *support for over-subscription of resources* (as was previously allowed internally within Cray [Pagel2009]).

Our expectations have further been raised by the *highly detailed accounting format* that SLURM offers out of the box: job accounting facilities of that granularity are currently missing in ALPS. The CLE 3.x mazama job database is under development and, since it is hosted on the SMW, it is restricted to system administrators. Short of parsing multiple ALPS logs in parallel, there is currently no non-tedious way of retrieving the same degree of *accounting detail* as provided by SLURM.

In conclusion, the cleanest and most future-proof solution to address the limitations we experienced would be to:

- open-source ALPS;
- replace (parts of) ALPS with SLURM;
- run slurmd rather than apsys on the compute nodes;
- use the SLURM/srun communication infrastructure for communication rather than the ALPS fanout tree.

At CSCS, we are interested in porting over some of the algorithms we have been using in the past years on the earlier Tcl scheduler. These include a streamlined backfilling strategy which achieved utilization of > 95% (averaged over 3 months).

In the longer term we would like to explore resource management options for other programming environments such as PGAS languages on a variety of architectures.

References

[Welch2003] Brent B. Welch, Ken Jones, Jeffrey Hobbs. "*Practical programming in Tcl and Tk*", 4th edition June 2003. ISBN: 0-13-038560-3.

[PBSPro2011] PBSPro Product webpages at http://www.pbsworks.com cited May 2011.

[Jette2003] Morris Jette, Mark Grondona. "*SLURM: Simple Linux Utility for Resource Management*", UCRL-MA-147996, Rev. 3, June 2003. Article submitted to the 2003 ClusterWorld Conference and Expo.

[Top500-2010] November 2010 edition of the Top500 web site http://www.top500.org/list/2010/11/100 cited May 2011.

[Karo2006] Michael Karo, Richard Lagerstrom, Marlys Kohnke, and Carl Albing. "*Application Level Placement Scheduler (ALPS)*". Paper presented at the 2006 Cray User Group meeting, Lugano, Switzerland.

[Albing2010] Carl Albing. "*ALPS, Topology, and Performance*". Paper presented at the 2010 Cray User Group meeting, Edinburgh, UK.

[cgroups2011] cgroups documentation at http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt cited May 2011.

[Pagel2009] Mark Pagel, Kim McMahon, David Knaak. "*Scaling the MPT Software on the Cray XT5 System and Other New Features*". Paper presented at the 2009 Cray User Group meeting, Atlanta, GA, USA.

[Maxwell2008] Don Maxwell. "*Restoring the CPA to CNL*". Slides presented at the 2008 Cray User Group meeting, Helsinki, Finland.

[Fahey2010] Mark Fahey, Nick Jones, and Bilel Hadri. "*The Automatic Library Tracking Database*". Paper presented at the 2010 Cray User Group meeting, Edinburgh, UK.

[Cray2010] Presentation titled "*Cray Corporate update*", slide 27. Slides available at http://www.nersc.gov/assets/Training-Materials/crayxtarchitecture.pdf cited May 2011.

Acknowledgments

We are indebted to the excellent support, encouragement, and expert help provided to us by the lead developers of SLURM: Morris Jette and Danny Auble of SchedMD, LLC. Their enthusiasm for the project and active engagement with the SLURM developer community are an inestimable advantage of SLURM compared to other resource management systems.

On the Cray side, we want to give our special thanks to Kongmanee Suvanphim, Jason Coverston, Benjamin Landsteiner, and Carl Albing.