Production Experiences with the Cray-Enabled TORQUE Resource Manager

Matt Ezell and Don Maxwell
High Performance Computing Operations
Oak Ridge National Laboratory
Oak Ridge, TN
{*ezellma,maxwellde*}*@ornl.gov*

David Beer
Senior Software Engineer
Adaptive Computing
Provo, UT
dbeer@adaptivecomputing.com

***Abstract*—High performance computing resources utilize batch systems to manage the user workload. Cray systems are uniquely different from typical clusters due to Cray's Application Level Placement Scheduler (ALPS). ALPS manages binary transfer, job launch and monitoring, and error handling. Batch systems require special support to integrate with ALPS using an XML protocol called BASIL.**

Previous versions of Adaptive Computing's TORQUE and Moab batch suite integrated with ALPS from within Moab, using PERL scripts to interface with BASIL. This would occasionally lead to problems when all the components would become unsynchronized. Version 4.1 of the TORQUE Resource Manager introduced new features that allow it to directly integrate with ALPS using BASIL. This paper describes production experiences at Oak Ridge National Laboratory using the new TORQUE software versions, as well as ongoing and future work to improve TORQUE.

***Keywords*-TORQUE; Resource Manager; Adaptive Computing; Cray; ALPS; Moab; HPC; Titan; Gaea**

I. Introduction

High performance computing resources utilize batch systems to manage the user workload. Job schedulers are designed to intelligently determine when jobs should run, optimizing for various goals such as high utilization or minimal queue wait. Resource managers typically accept job submissions and handle job launch. Cray systems are uniquely different from typical clusters due to an additional layer called Cray's Application Level Placement Scheduler (ALPS). ALPS manages binary transfer, job launch and monitoring, and error handling. Batch systems require special support to integrate with ALPS using an XML protocol called BASIL.

Previous versions of Adaptive Computing's TORQUE and Moab batch suite integrated with ALPS from within Moab, using PERL scripts to interface with BASIL. This would occasionally lead to problems when all the components would become unsynchronized. Additionally, TORQUE was unaware of the Cray compute nodes. Version 4.1 of the TORQUE Resource Manager introduced new features that allow it to directly integrate with ALPS using the BASIL protocol.

This paper describes early experiences with the newest versions of the TORQUE resource manager. Early on, software bugs related to the newly-introduced multithreading features prevented successful deployment of the new versions of TORQUE. Through close collaboration with Adaptive Computing, the software improved significantly to the point where it was acceptable for use on the Titan and Gaea systems. Additionally, this paper describes production experiences at Oak Ridge National Laboratory using the new TORQUE software versions and describes future collaborative work to improve Cray-enabled TORQUE.

II. Cray Application Level Placement Scheduler (ALPS)

The term "resource manager" is overloaded in the batch processing world, and the combination of systems needed to effectively control batch processing on the Cray X-series platform certainly supports the confusion. At the lowest level in the Cray hierarchy sits the Cray Application Level Placement Scheduler, commonly referred to as ALPS [1]. While in most instances, a batch system is making scheduling decisions based upon configuration of center policy, ALPS is the piece of software at the lowest level that is handed information from the batch system to ultimately launch the job onto the compute nodes.

Not only does ALPS launch jobs, it maintains compute node state and reservations to manage job placement and resource utilization. Through a series of daemons that typically run on the Cray boot or system database (sdb) node, ALPS imports hardware configuration information from the system database to provide memory, CPU and GPU resources available on each compute node. Given a heterogeneous system, this would be essential to providing the user with a mechanism to request the resources needed to run a particular application. Each compute node also has a state and mode associated with it that informs ALPS whether the node is up or down and whether it is in interactive or batch mode. Up and down states are self-explanatory, and there are a few other states that will not be discussed that are in the end classified as up or down, but interactive or batch mode requires some explanation.

ALPS can operate without a batch system sitting at a higher level providing information. This is called interactive mode, and it is basically a FIFO queue that requires users to

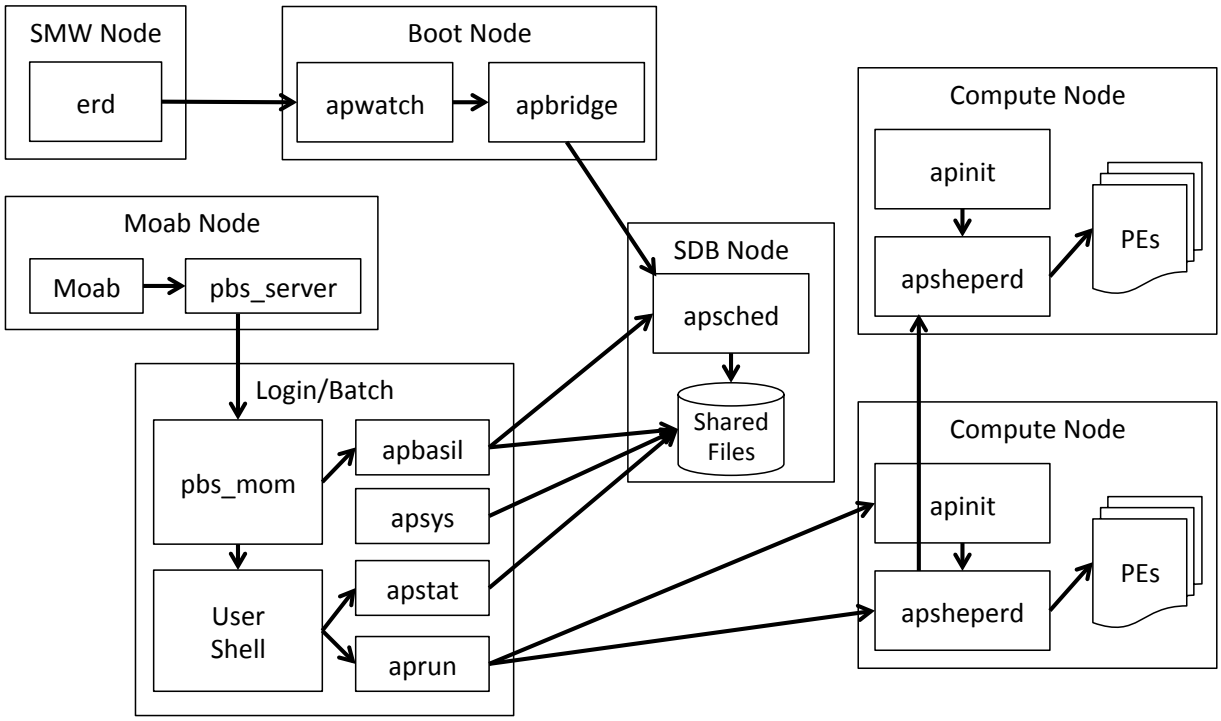

Figure 1. High-Level ALPS Design

run ALPS commands that sit and wait for available resources to run jobs. How is this different from a batch system? In a nutshell, batch systems provide a mechanism for the user to submit a job to be run at a later time without the burden of making sure the machine doesn't reboot taking the waiting ALPS command down with it. Batch systems also allow a priority-based reordering of jobs based on policies determined by the center. In contrast, batch mode simply enables the use of a batch system for job launch ignoring any user-supplied ALPS commands run outside of the batch system.

Reservations are used to manage availability of nodes that are in an up state. When a job is launched, it is assigned to a set of nodes. Those nodes are exclusively reserved for that job. When the job finishes, the reservation is destroyed, and those nodes are available for the next job. Reservations are simply the mechanism by which a job receives exclusive access to the resources necessary to run the job.

All of the ALPS information must somehow get communicated to the batch system in order to provide the user with the available resources on the system and to maintain a consistent state for nodes, reservations, and jobs. ALPS provides an API called *apbasil* - BASIL being an acronym for Batch Application Scheduler Interface Layer. BASIL is an XML-based protocol that provides batch systems with the ability to retrieve inventories of compute nodes, their states and reservations, and the ability to create, confirm, and delete reservations. Using this interface, batch systems are able to manage all aspects of job submission, scheduling, placement and deletion by communicating with ALPS which communicates directly with the compute nodes.

III. MOAB-ALPS INTEGRATION DESIGN

Prior to TORQUE version 4.1, the primary interaction between the batch system and ALPS was managed by Moab. Moab has the concept of a "native" interface that allows external scripts to handle logic in user-configurable ways. For Cray systems, Moab's interaction with the TORQUE resource manager and ALPS was setup as a native interface. Multiple scripts were provided to manage resource inventories, ALPS reservations, and job launch:

node.query.xt4.pl
: This script queries BASIL to get the configuration and state of all compute nodes on the system

job.query.xt4.pl
: The job query script executes and parses TORQUE commands to obtain a list of all jobs known to the system, including details such as state, owner, account, queue, and resource requests.

partition.query.xt4.pl
: The partition query script talks to both ALPS and TORQUE to determine what ALPS reservations exist and if they correspond to a running job

partition.create.xt4.pl
: This script interfaces with BASIL to create an

ALPS reservation based on the resource requests of a job. It does not confirm the request, as this must be done by TORQUE once the SID or PAGG is known

job.start.xt4.pl

This script determines the best pbs_mom to host the job and forces execution on this node

partition.delete.xt4.pl

The partition delete script, as its name implies, is used to remove ALPS reservations once jobs have completed

Once TORQUE is signalled to start a job, it must spawn a child process to run the job script. If enabled, it will use the "job" interface to create a Process Aggregate (PAGG) container for the process. TORQUE must then "confirm" the ALPS reservation by supplying the PID or PAGG of the job. ALPS is then able to use this information to distinguish which processes belong to which reservation.

In this setup, TORQUE is only configured to know about the batch nodes that are running the pbs_mom daemon. TORQUE is completely unaware of the compute nodes. While this avoids some complexity, it obfuscates the process a bit. All logging from TORQUE only knows about the batch node used to start the job, not the computes nodes on which the job was running. This could be problematic if TORQUE accounting logs are used extensively.

Additionally, Moab must spend extra time performing a "fork" and "exec" to run the perl scripts. It must then process the string-based output to turn it into data structures that can be ingested by Moab's internal processing. This can lengthen Moab's average iteration duration, which has a negative impact on interactive use of Moab's command line tools.

An issue would occasionally arise on production systems that had a very negative impact on utilization. Certain circumstances could cause an ALPS reservation to remain after the job had exited. These "orphan" reservations would block system resources until manually removed. In an attempt to avoid this, Moab added code to help find and destroy these orphan reservations. By setting the **MOABPARCLEANUP** environment variable, Moab would know to try to remove reservations that it didn't expect to be present at the expense of extra processing each iteration.

IV. New Design

The new TORQUE-ALPS design greatly simplifies the implementation by reducing integration points and coupling things where they naturally make sense. The minutiae of creating, confirming, and releasing jobs doesn't need to be performed by a scheduler; moving this all to TORQUE daemons allows Moab to do what it does best schedule. TORQUE daemons already manage everything that has to do with setting up jobs, starting them, and cleaning up after them once completed, so the ALPS interactions naturally belong here also. Finally, having all of the integration a single daemon reduces integration points, making the code easier to understand and maintain.

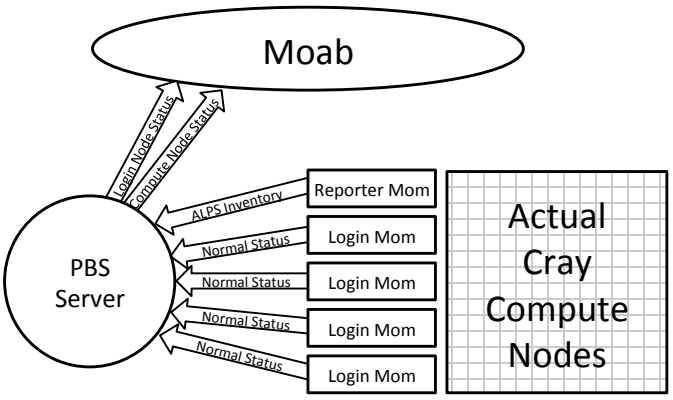

Figure 2. New Reporting Structure

A. How It Works

All ALPS integration now takes place on the pbs_mom daemons as shown in Figure 2. Two different kinds of pbs_mom daemons are run for this setup: a reporter mom and one or more alps_login moms. The reporter mom's only function is to report the inventory information to pbs_server, which then discovers all of the compute nodes.

The other mom daemon is the alps_login type, which manages job starts. The login mom creates and confirms the reservations for the job before launching it. When the job is done, the mom releases its ALPS reservation. The job start process is diagramed in Figure 3.

Moab no longer knows anything about ALPS. From Moab's perspective, it is scheduling two different partitions (clusters) - the login nodes in one partition and the compute nodes in the other. Moab is only making scheduling decisions, and so it is now only given information relevant to scheduling. This is outlined in Figure 2. Moab maintains the ability to schedule login-only jobs for the purpose of compilation, data transfer, or other simple, short tasks.

The pbs_server ties the two together. When a node status is given, the reporter mom's status is presented as the status of all of the compute nodes, and the logins' statuses are given as any other pbs_mom's status appears. When Moab runs a job, it only selects which compute nodes that job will use, and pbs_server uses a round robin method to select which login daemon should be responsible for starting that job.

B. Advantages

The new design simplifies configuration. No special binaries for Moab, TORQUE's server or mom daemons are required, not even for the distinct types of mom daemons. Less configuration makes for easier setup.

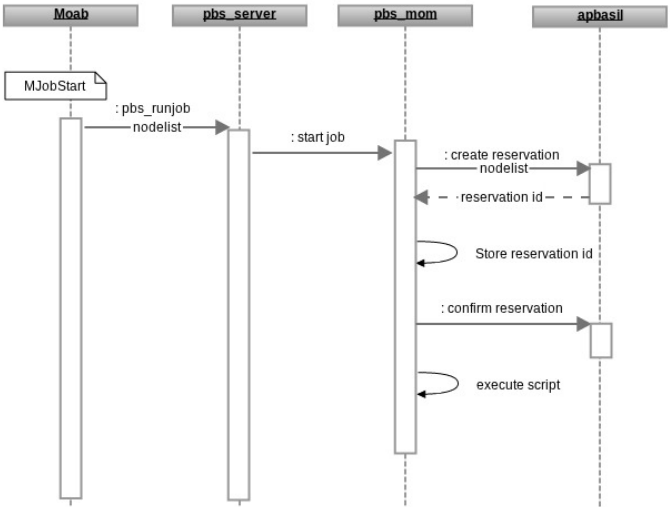

Figure 3. Job Starting

The whole model inherits superior support from Moab because from Moab's perspective, scheduling the Cray is the same as scheduling any other cluster: it is no longer a one-off. Additionally, the code path that was used for parsing output from the scripts in significantly slower than the code path for interfacing with TORQUE for two reasons: text parsing happens faster and the interaction is now through an API instead of a fork/pipe model.

Moab and pbs_server can now be moved outside of the Cray, allowing a number of benefits. You can install them on as powerful of nodes as you like, and aren't bound to whatever you have already. If, after having the machine for a while, you decide you need to upgrade the server and scheduler nodes then you can do this without upgrading the entire machine. This also gives you back whatever resources on the Cray Moab and pbs_server were occupying.

V. Production Experiences

After initial conversations with Adaptive Computing discussing the transition of ALPS interactions from Moab to TORQUE to provide better ALPS reservation synchronization, an initial beta test at scale was undertaken on site at ORNL in June 2012. The test was performed using Jaguar and was fairly successful, but, as expected, pointed out a few modifications that were needed. Some minor modifications were needed to clean up reservations correctly with some additional tweaks needed to support interactive jobs. We continued the debugging cycle by running the new batch architecture on two of our Cray test and development systems (TDS) at ORNL.

A. Early Experiences on Gaea

The National Climate-Computing Research Center (NCRC) at Oak Ridge National Laboratory houses and operates high performance computing resources for the National Atmospheric and Oceanic Administration (NOAA) and its research partners. NCRC's flagship system is named Gaea, after the Greek goddess of earth. Gaea was delivered and upgraded in phases, ultimately culminating in two production partitions with a combined peak performance of 1.1 petaflops.

In July of 2012, the "c1" partition of Gaea was upgraded from a Cray XT6 system with the SeaStar interconnect and AMD "Magny Cours" processors to a Cray XE6 system with the Gemini interconnect and AMD "Interlagos" processors. The previous system used TORQUE 2.5.x. Moab was setup in a grid; there was a central Moab that made scheduling decisions and an instance of Moab on c1 that interfaced with ALPS. The hardware upgrade also required an operating system software upgrade, and it was determined that TORQUE version 4.1 should be installed and tested.

At the time, version 4.1.0 was the latest release available. Several important bugs had already been identified and fixed, so ORNL decided to pull souce from the 4.1-fixes branch in subversion. Installation and initial testing were successful, but it wasn't long before problems were discovered.

The first problem encountered in the acceptance test was missing *PBS_O_** environment variables. The acceptance harness used variables such as *PBS_O_WORKDIR* to setup the correct environment prior to running a job. When those were missing, the jobs would fail.

The most severe bug found early on was related to improper handling of environment variables. When variables were set without values, the qsub command would segfault and fail to submit the job. Also, variable values that contained comma or newline characters would be incorrectly split and sent to the environment as multiple variables. This behavior was not obvious at first glance, but it was causing an acceptance application to fail due to a missing environment variable that increased the thread stack size.

TORQUE 4.0 included a major architectural change: the pbs_server was made multi-threaded. While this is essential for performance and scaling, it also introduces the possibility of race conditions and deadlocks. Although the developers were careful in their implementation, several deadlocks were experienced.

Another bug caused Cray jobs to fail when the pbs_server was restarted. It was due to Cray nodes not being present when the jobs were being recovered. X11 forwarding in interactive jobs was not working correctly, preventing the use of debuggers and GUI frontends. A problem was also encountered where trying to delete a running job in the Cray environment failed. The server was trying to communicate with the cray compute node instead of the login node that was running the job script.

As each major bug impacting system operation was identified and fixed, ORNL pulled the latest source from subversion and ran with that. TORQUE version 4.1.1 was released on August 30, 2012, but there were no significant

relevant fixes above the version already in production. In late October, the Gaea environment moved to TORQUE version 4.1.3. This version contained several fixes, including several patches developed at ORNL.

B. Experiences on Titan

While Jaguar never ran the new architecture aside from the beta test shot, Titan began its life in September 2012 running the new design. While many of the issues inherent in a new design had already been discovered and fixed during the beta, subsequent use on the TDSs, and production on Gaea; the Titan acceptance team put the new software stack through its paces at scale for true production. This transition also provided a new opportunity to externalize both Moab and TORQUE servers from the Cray platform to provide job access for users even when the Cray was unavailable. Two fundamental changes at once is generally not a good idea, but all in all, things have went fairly well.

Overwhelmingly, the primary problem that has been seen on Titan is deadlocks in the TORQUE server. Threading the version 4 TORQUE server is clearly a step in the right direction, but it has come with some growing pains. A deadlocked TORQUE server causes issues for the entire batch system from simple job submission failure to a hung or at least very slow Moab server. Primarily due to the fact that the end users were seeing job submission failures, a script was created early in the acceptance period which first determined the TORQUE server was deadlocked, gdb attached to the server and generated a core file, and then restarted the server. By running this script on a regular schedule via cron, the pain felt by the users became much more bearable until the deadlocks were found and fixed. Through the efforts of both ORNL and Adaptive staff, all known deadlocks have been fixed, and the batch system is running well at this point.

The transition to an external Moab and TORQUE server has certainly been well received by the users. Having the ability to manipulate jobs when the Cray is unavailable provides the user with everything needed for the job process except the actual execution. Some effort had to be devoted to finding a TORQUE server bug that prevented this from working, but that has now been fixed as well.

Again, the transition to the new architecture and the external servers has been a success in spite of a few growing pains. Ultimately, the users have seen benefits from both better synchronization with ALPS and the ability to manipulate jobs while the Cray itself is unavailable.

VI. Future Work

Work is ongoing to improve the quality of TORQUE. Unit testing coverage is improving, and it will continue to improve over time. The TORQUE 4.2 series moved from a C compiler to a C++ compiler, giving access to additional language constructs as well as the Standard Template Library (STL). Over time, some of the one-off TORQUE implementations of standard data structures will be replaced with their counterparts from the STL. Work is ongoing to improve large job start time and implement hostlist compression. Additionally, there will be improvement on the Cray-specific features. Some specific ideas include:

- Additional logging of interactions between TORQUE and ALPS, including full XML at high log levels
- Taking advantage of new features in higher ALPS BASIL versions, such as inventory *changecount* and reservation release claim count. Version 1.3 supports additional granularity for thread placement.
- More robust handling of reservation IDs and orphan handling

VII. Conclusion

TORQUE's quality has been somewhat spotty over the last year as major architectural changes have been implemented. Thanks to the diligent work of the TORQUE developers and the community input, TORQUE has significantly improved.

Cray's ALPS software is very unique, and it requires batch systems to add special support to interoperate. Adaptive's original design that interfaced with ALPS from Moab was effective, but the new TORQUE-based ALPS integration is more straightforward and higher performing. The new software provides benefits that are highly visible to end-users.

Acknowledgment

The authors would like to thank the TORQUE developers as well as the TORQUE community for constantly improving TORQUE.

References

[1] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing, "The application level placement scheduler," *Cray User Group*, pp. 1–7, 2006.