The Dynamic PBS Scheduler

Jim Glidewell, Boeing Information Technology

ABSTRACT: *Defining multiple queues can help a site to control the mix of system resources consumed by running jobs. Unfortunately, static limits associated with such queues can lead to underutilization of the overall system. This paper will describe our method for adjusting queue limits dynamically, based on the priority and resource requirements of the current mix of jobs.*

Our HPC Environment

Boeing's Puget Sound High Performance Computing (HPC) environment consists of a Cray X1 with 128 MSPs (Multi-Streaming Processors) and 1 terabyte of main memory, and a number of Linux clusters.

There are seven Linux clusters currently: two 128 node clusters, three 256 node dual-Opteron clusters, and a single 512 node cluster. All of the systems are dual-Opteron and use Myrinet as their MPI communication fabric, and all nodes have Gigabit Ethernet connections for TCP traffic.

The Cray X1 and Linux cluster nodes share access to a single, Panasas storage system, which serves as primary permanent storage for the clusters, as well as shared common space between the two types of systems.

Background and Motivation

Our site has long used multiple queues to give us more fine-grained control over the mix of jobs running at any one time on our systems. While multiple queues are useful, they have drawbacks when presented with a work mix that varies over time. The most typical problems are poor utilization of system resources and excess queue wait time for user jobs. System resources can go to waste when a restrictive queue limit holds back waiting user jobs, even though resources are available. Such a situation also needlessly elongates user turnaround times. The fixed queue limits that allow the system administrator to reserve some resources for a specific queue can cause significant waste if the workload changes and no longer matches previous characteristics.

In practice, we saw this problem often when trying to reserve a limited number of MSPs on our Cray X1 for short-lived, minimal resource jobs. When using a fixed limit, we often found we were either reserving too many, or not enough, nodes for this class of jobs, resulting in wasted resources or unhappy users.

We needed a more flexible method of apportioning the system between user groups and job types that fixed limits on queue could provide. To do this, we created a daemon that would serve to adjust the queue limits to reflect the current work mix on the system. This daemon is referred to as the Dynamic PBS Scheduler, dyn_pbs.

History

We have long used dynamic scheduling techniques to supplement the basic scheduler incorporated into the batch subsystems of our HPC systems. Our first dynamic scheduler daemon was written in "C" and was deployed on our Cray T90 systems starting in the mid-1990's. We used that daemon to control the NQS (Network Queuing System) queues and queue complexes to try to optimize the mix of job memory sizes and CPU limits, with the goals of controlling the overall memory subscription of the system and providing good turnaround for smaller jobs. The daemon was table-driven, with weights, targets, and limits for queues and complexes.

We used a much more limited and less flexible scheduling daemon on our SGI Origin 3800 system. This daemon was written in Perl, and did some very basic modification to PBS (Portable Batch System) system and queue limits based on the current workload.

The design of the Dynamic PBS Scheduler (dyn_pbs) for our Cray X1 was a return to the table-based approach of the original dynamic scheduler, but coded in Perl for faster deployment and ease of maintenance and enhancement. It controls the PBS Professional batch queuing system installed on our Cray X1.

Goals

Our goals in creating the Dynamic PBS Scheduler were functionality, simplicity, and safety. From a functional point of view, we wished to provide reasonable turnaround for all job classes, honor user-specified priorities, minimize the amount of wasted system resources, and meet some broad throughput target provided to us by our customers. We wanted a system that was fairly simple, which would require a modest amount of development effort, and would be easy to adjust and tune as requirements changed. Finally, it was critically important that the daemon be "safe" – it should not do anything which might compromise the throughput of the system.

To meet the goal of safety, the daemon was designed to do the minimum changes necessary and to run as an ordinary user (with PBS administrator rights). As a fail-safe in case of daemon failure, and automated process run hourly from the Unix cron facility is used to check for the presence of the daemon and re-launch it if necessary.

Workload Characteristics

Our workload is less diverse than many other Cray sites, but it consists of a small set of job types with radically different resource requirements. The bulk of our workload

consists of jobs requiring 16 MSPs, which can run for days at a time. At the other end of the spectrum, we have a large number of very small and short running jobs that require immediate turnaround. Between these two extremes are a mix of job sizes and characteristics.

The bulk of our workload is CFD (Computational Fluid Dynamics) primarily using the Overflow code originally authored at NASA. Overflow is used both for single case analysis, as well as for optimization runs where the Overflow application is run multiple times to optimize a specific CFD characteristic. We also see moderate use of Tranair, a Boeing-developed CFD code for aerodynamic analysis. Our small, fast-turnaround jobs consist primarily of structural analysis jobs running ATLAS, a Boeing-developed structural analysis code.

Overview of the Dynamic PBS Scheduler

The basic design of the PBS Dynamic Scheduler is to do the following: get the status of all jobs (queued and executing) from PBS, compute new limits for the managed PBS queues, direct PBS to update the limits, and repeat after a fixed delay.

We use a simple "qstat –f" command to extract job status detail from PBS. We have used the PBS API (Application Programming Interface) for other tools, but given the default cycle period of five minutes for dyn_pbs, using qstat output seemed simpler, and allowed us to simulate various conditions by simply replacing the qstat text output with an edited test file.

The computation of new limits is based on tables that specify a variety of weights and limits for the server, queues, and jobs. Each job contributes its "weight" to the queue that contains it.

These total queue weights are then used to determine what the proper number of MSPs to allocate to each queue should be, in consultation with the limits defined in the dyn_pbs tables. These new MSP limits are then fed to PBS using the appropriate qmgr (queue manager) directives.

Weight and Limits

The following table lists the weights and limits that the Dynamic PBS Scheduler uses to determine the proper queue limits:

Domain	Limit or Weight
Server	Server subscription factor
Server	Reserved MSPs
Queue	Minimum number of MSPs
Queue	Default number of MSPs
Queue	Maximum number of MSPs

Queue	Queue weighting factor (for queued jobs)
Queue	Queue additional MSP weight per running job
Queue	Queue maximum weight
Queue	Queue oversubscription factor
Queue	Queue oversubscription order
Job	Priority-based MSP weighting factor
Job	Priority-based job weight

Table 1. Weights and limits associated with PBS objects

Computing Queue Limits

Computation of the individual queue limits starts by summing the weights of all member jobs in that queue. Jobs carry different weights based on their priority, their current state (running or queued), the number of SSPs or MSPs they require, and the weighting factors associated with the queue to which they belong.

Once the total weights are computed (and possibly reduced based on a queue's maximum weight), an initial division of the systems MSPs is made based on the weight of each queue. If any queue exceed their maximum number of MSPs, those MSPs are returned to the pool and reallocated based on the previous apportionment.

One issue that arises when there is a shift in the mix of queued and executing jobs is that after the limits are computed, some queues may have a limit lower than the number of MSPs currently being used in that queue. Ignoring this situation will result in oversubscription of the machine, which often results in the last MSP being consumed, leaving no resources for the small, turnaround-sensitive jobs. To avoid this situation, the number of MSPs that would exceed the new limit is multiplied by an oversubscription factor, then the resulting MSP pool is subtracted from all the computed limits proportionally. This temporarily reduces the available MSPs until sufficient resources are available to meet the targets based on the current work mix, and allows a gradual migration to the target state.

After all computations are complete, and the new MSP limits have been established, the current computed limit for each queue is compared to the previous value. If the value for a specific queue has changed, a qmgr directive is issued for that queue. By only issuing changes when needed, we avoid unnecessarily flooding the PBS and dyn_pbs logs with repetitive messages and reduce the amount of traffic that the PBS server needs to handle.

Experience

During early production on our Cray X1, the issues of trying to balance the needs of the users with large, long running jobs with those of the small jobs were somewhat daunting. Complaints from members of the latter user community were fairly frequent, and active

monitoring of PBS was a daily requirement. The need for some method of adjusting queue limits based on current workload was apparent fairly early on.

We began implementation of the initial version of dyn_pbs roughly six months after we took delivery of the Cray X1. Since that time, it has seen several enhancements, including distribution of MSPs beyond fixed queue limits, improving the handling of over-subscribed queues, and ensuring that the number of production MSPs is kept current. Our current version of dyn_pbs has been running without further enhancements since late 2006.

During our early usage of dyn_pbs, tuning of the table entries for weights and limits was fairly frequent, but that activity has become very infrequent. We have not needed to change the limits tables for several months, despite some fairly large swings in workload mixes.

A scheduler that does not require daily system administrator attention, and which provides users sufficiently reliable turnaround such that the user support phone does not ring of a frequent basis is a successful one. dyn_pbs has significantly increased user satisfaction and reduced administrative workload on our Cray X1.

Measuring Success

The two primary goals of the Dynamic PBS Scheduler were maximizing system utilization, while minimizing turnaround time for users. The measurement of system utilization is relatively straightforward, since there are a number of ways to view resource reservation and utilization on the Cray X1. As we intended, dyn_pbs reduced the number of cases where resources were available but the queue limits prevented a job from running, and overall system utilization is quite good.

The more difficult item to quantify is user turnaround experience. We have looked at several metrics, such as average wait time, average wait-to-run ratio, both unweighted and weighted by the number of MSPs (or nodes on our cluster systems). We also tried bucketing jobs by size, and looking at various percentiles – none of them gave us a metric that seemed to reflect our "gut feel" of when things were running well versus when we knew we had a significant backlog and users were unhappy.

One issue we have seen with studying turnaround is that *outliers count*. Single jobs that don't run for days (or weeks) may not have a big effect on the averages, but can significantly impact an engineer's ability to get his job done. Likewise, user behavior like submitting hundreds of jobs at one time can skew the average turnaround numbers, while not really impacting the user (who may be well aware that the work will run over the next month). Summary statistics did not really seem to tell us what we needed to know, so we started to look for a way to present the data that would give us a "bird's eye view" of job turnaround.

Turnaround Plots

Since summary data wasn't telling us what we wanted to know, we decided to see if we could somehow fit all of the job detail history for a day, week, or month onto a single page plot. In pursuit of this goal, we began development on what came to be known as the "red/green" charts.

These charts are based on each job being represented by a pair of rectangles, with the red rectangle representing wait (or queued) time, and the green rectangle representing the jobs run time. The height of the two rectangles represents that number of SSPs (or nodes) a job is reserving. The job's rectangle pair is placed along the horizontal axis, which represents a time line. Vertical placement of the job is not significant, and is based on first-fit where the rectangle will not overlap any previously placed rectangles.

By using some rather simple heuristics (including sorting jobs by submit time and keeping track of the rightmost filled pixel on every horizontal line), we are able to quickly create plots which display hundreds or thousands of individual jobs and their respective wait and run times.

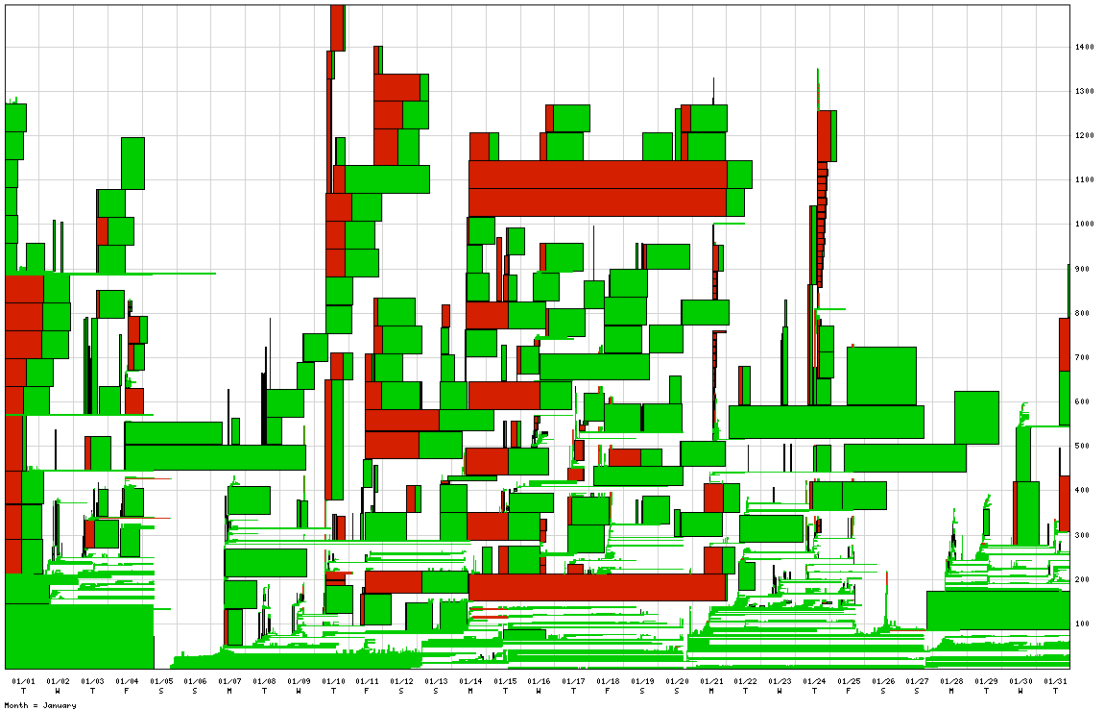

Figure 1. Example of "red/green" chart

The red/green charts contain a large amount of detail, and can often provide both answers and areas for further investigation with minimal study. Periods of heavy demand and cases where a single user injects a large number of jobs at one time are relatively easy to identify. Jobs with long wait times, and short run times, are worthy of looking at in detail.

Simply observing the overall ratio of green to red gives a good indication of whether the system is lacking sufficient capacity to meet current engineering demands.

These charts have been a useful tool in analyzing job turnaround on both our Linux clusters and on the Cray X1.

Summary and Conclusions

While using multiple batch queues allow finer administrative control over job selection and execution, they can impact both user turnaround and overall system throughput if the job mix varies significantly over time. By adding dynamic adjustments to our queue limits, using the Dynamic PBS Scheduler, we have been able to better address user turnaround requirements and maintain overall high system utilization.

About the Author

Jim Glidewell has been a member of Boeing's HPC group for over twenty years, working on a variety of systems from Cray, SGI, CDC, and others. He is currently serving as the CUG User Support SIG Chair. He can be reached at The Boeing Company, P.O. Box 3707 MC 7J-04, Seattle WA 98124-2207; E-mail: james.glidewell@boeing.com