Using Quality of Service for Scheduling on Cray XT Systems

Troy Baer, *National Institute for Computational Sciences*

ABSTRACT: *The University of Tennessee's National Institute for Computational Sciences (NICS) operates two Cray XT systems for the U.S. National Science Foundation (NSF): Kraken, an 88-cabinet XT5 system, and Athena, a 48-cabinet XT4 system. Access to Kraken is allocated through the NSF's Teragrid allocations process, while Athena is currently being dedicated to individual projects on a quarterly basis; as a result, the two systems have somewhat different scheduling goals. However, user projects on both systems have sometimes required the use of quality of service (QoS) levels for scheduling of certain sets of jobs. We will present case studies of three situations where QoS levels were used to fulfill specific requirements: two on Kraken in fully allocated production service, and one on Athena while dedicated to an individual project. These case studies will include lessons learned about impact on other users and unintended side effects.*

KEYWORDS: XT4, XT5, scheduling, quality of service, QoS

1. Introduction

The University of Tennessee's National Institute for Computational Sciences (NICS) operates two Cray XT systems for the U.S. National Science Foundation (NSF): Kraken, an 88-cabinet XT5 system, and Athena, a 40-cabinet XT4 system. Access to Kraken is allocated through the NSF's Teragrid allocations process, while Athena is currently being dedicated to individual projects on a quarterly basis. The details of the two systems are shown below.

System	Kraken	Athena
Cabinets	88	48
Compute Nodes	8,256	4,512
Processor	AMD Opteron 2.6 GHz hex-core	AMD Opteron 2.3 GHz quad-core
Total Cores	99,072	18,048
Peak Performance (TFLOP/s)	1,030.3	165.9
Memory (TB)	129.0	17.6
Disk (TB)	2,400.0	85.0
Disk Bandwidth (GB/s)	30.0	12.0

2. Scheduling on XT Systems with TORQUE and Moab

Like many large XT systems, Kraken and Athena use the TORQUE batch environment. TORQUE [2] is an open source resource management software package with a long and venerable history, being derived from PBS and used on a large number of high performance computing systems over the years. Like all PBS variants, TORQUE is modular, consisting of three components: a queue server daemon (`pbs_server`), a scheduler daemon, and a "machine-oriented mini-server" daemon (`pbs_mom`). A typical TORQUE installation has one `pbs_server` instance, one scheduler instance, and as many `pbs_mom`s as there are compute nodes. However, this arrangement changes slightly on Cray XT systems, as it is impractical to run a `pbs_mom` on every XT compute node; instead, `pbs_mom`s run on a small set of dedicated service nodes, and jobs run on these service nodes spawn processes on the XT compute nodes by invoking the `aprun` command of the Cray Application-Level Placement Scheduler (ALPS) service [11].

The interface between the TORQUE `pbs_server` and scheduler is well defined, and while TORQUE includes a simple scheduler (`pbs_sched`), most sites choose to use one of the many available third-party schedulers. Like many large XT systems, Kraken and Athena both use the Moab scheduler. Moab [3] is an extremely powerful and flexible commercial scheduler software package that supports a wide variety of batch environments, including all PBS variants, LSF, LoadLeveler, and SLURM. Moab also supports a number of advanced scheduling capabilities such as advance reservations, quality of service (QoS) levels, consumable resource management, and a highly configurable priority and policy engine. On Cray XT systems, Moab communicates with ALPS as well as TORQUE, which is accomplished by interfacing to a native resource manager, a set of “glue layer” scripts that sit on top of ALPS and TORQUE services.

2.1 Moab Scheduling Algorithm

At Moab's core is a seven-stage scheduling algorithm [6], which generally runs at regular intervals but can also be provoked by external events such as job submissions. The seven steps of the Moab scheduler iteration are as follows:

1. Update status information from resource manager(s)
2. Refresh reservations
3. Start jobs with reservations (if possible)
4. Start jobs with highest priority (if possible)
5. Backfill jobs
6. Update statistics
7. Handle client requests

On Cray XT systems such as Kraken and Athena, steps 1, 3, 4, and 5 require interactions with TORQUE and ALPS through the XT native resource manager interface.

2.2 Queue Structure

To simplify management of Kraken and Athena, the two systems share a common queue structure, which differs only in core count ranges between the two machines. This structure is intended to be largely invisible to users, who generally submit jobs to one of two routing queues (`batch` and `debug`) that filter jobs into the actual queue based on size and time requests.

Queue	Max. Walltime	Max. Cores (Athena)	Max. Cores (Kraken)
`small`	24 hours	512	512
`longsmall`	60 hours	512	256
`medium`	24 hours	2,048	8,192
`large`	24 hours	8,192	49,536
`capability`	24 hours	18,048	99,072
`dmover`	24 hours	0	0
`hpss`	24 hours	0	0

The `dmover` and `hpss` queues are present to facilitate scheduled data transfers using GridFTP and HPSS, respectively.

2.3 Job Priorization

As with the queue structure, Kraken and Athena share a common job prioritization model to simplify management. Job priorities on Kraken and Athena are based primarily on the number of cores requested. Secondary factors used in the priority calculation include the job’s queue time and its expansion factor [10], although recently the latter has been removed because it often caused unexpected effects.

2.4 Quality of Service Levels

In Moab, a *quality of service level* is an object enabling a job to request special scheduling considerations, which may include modified priority or throttling policies, different behavior with respect to reservations or backfill, and the ability to preempt or be preempted by other users' jobs [8]. A QOS level may be applied to a job in one of two ways, either by automatic assignment by Moab or by an explicit request as part of the batch job (e.g. `qsub -l qos=foo` in TORQUE). In either case, some credential associated with the job must be able to access a QOS level in order for the job to use that QOS; these credentials include users, groups, accounts/projects, and queues/classes.

In normal operation on Kraken and Athena, there were originally only two non-default QOS levels in use, which all users and projects were (and are) able to access. One of these QOS levels, `sizezero`, is applied at the Moab level using a job template and is rarely (if ever) explicitly requested by users. The `sizezero` QOS causes `size=0` jobs, which use no compute nodes and are typically data movement jobs in the `dmover` and `hpss` queues, to run in a timely manner despite their relatively low priority by applying the `RUNNOW` flag to them. The `sizezero` QOS also limits how many `size=0` jobs can run concurrently, to try to avoid overwhelming the aprun and login nodes where these jobs run. The second QOS level used in normal Kraken operations is `negbal`, which is applied automatically by the TORQUE submit filter to jobs from projects whose allocations have negative balances. The `negbal` QOS applies a large negative priority modifier to jobs which request it, so that jobs from projects with negative balances do not run unless those cycles would otherwise go unused. However, as will be seen below, a number of other QOS levels were added over time to deal with various situations and requirements.

3. Case Studies

3.1 Normal Operation on Kraken

To establish a baseline case, the following table summarizes the queue times observed over a roughly six-month period of Kraken's production lifetime from 5 Oct 2009 to 31 Mar 2010, during which the overall system utilization was 65.62%:

QOS	Jobs	CPU Hours	Min Queue Time	Max Queue Time	Mean Queue Time
default	220,197	241.7M	00:00:03	858:59:59	03:59:26
negbal	13,137	39.5M	00:00:04	398:04:00	08:51:31
sizezero	2,231	0.0M	00:00:05	262:51:39	04:11:42

Unsurprisingly, jobs with the `negbal` QOS wait considerably longer between job submission and job start than jobs with other QOSes. However, jobs with the `sizezero` QOS have slightly worse queue wait times than those with the default QOS. This is likely due to the fact that the `sizezero` QOS did not initially include the `RUNNOW` flag; this was a later addition in response to `sizezero` jobs waiting longer than desired without that flag's use.

3.2 Nightly Weather Forecasting on Kraken

The Oklahoma University's Center for the Analysis and Prediction of Storms (CAPS) was given a Teragrid allocation on Kraken to run part of their 2009 Spring Forecast Experiment, which ran from 16 Apr 2009 to 12 June 2009 [1]. This required nightly standing reservations on Kraken of approximately 10,000 cores for 8 hours on five days per week, Sunday through Thursday. Of those 10,000 cores, 9,600 cores were used for a WRF-based forecast simulation [9], 320 cores were used for post-processing analyses that ran concurrently to the forecast, and 80 cores were held in reserve in case node failures forced a forecast to be restarted. These reservations started at 10:30pm at night and went until 6:30am the following morning. The forecast and post-processing reservations were managed separately, as the post-processing jobs did not start until about half an hour after the forecast job started and sometimes continued for an hour or more after the forecast job ended. Further complicating this was the fact that Moab does not currently support standing reservations with periodicity of a day that start on one day and end on another, so each “reservation” actually consisted of two separate component reservations, a PM component running from 10:30PM to midnight and an AM component running from midnight to 6:30AM. Access to these reservations were managed by requesting one of two QOS levels, `capsforecast` and `capspostproc`, which could only be accessed by members of the CAPS project. These QOSes also had increased priority so that they would start as soon as possible when the reserved resources were available.

The following table summarizes the queue times observed during the period of the CAPS Spring Experiment's use of Kraken from 16 April 2009 to 12 May 2009, during which the overall system utilization was 66.02%:

QOS	Jobs	CPU Hours	Min Queue Time	Max Queue Time	Mean Queue Time
default	35,257	55.6M	00:00:02	466:24:39	03:51:50
negbal	2,692	3.0M	00:00:04	146:53:14	02:15:37
sizezero	611	0.0M	00:00:03	132:26:36	01:04:35
capsforecast	68	3.0M	00:00:05	42:01:04	01:48:31
capspostproc	2,155	0.1M	00:00:03	26:42:12	00:02:49

As can be seen above, CAPS jobs were run considerably more quickly than regular jobs; in fact, the largest component of queue time for CAPS forecasting jobs was that they were generally submitted around 9PM with a flag that prevented them from being eligible to run before 10:30PM. Surprisingly, jobs with the `negbal` QOS actually received better service than regular jobs did in terms of queue time, as these jobs are tend to be small and short, making them easy to backfill around the reservations required by the CAPS project.

However, service improvements for the CAPS

project came at a cost in service to other users, particularly those who had large or long running jobs. Jobs in the `longsmall` queue saw an average queue time of over 96.5 hours, whereas during the period previously discussed, similar jobs saw an average queue time of slightly less than 52 hours. Users with `capability` jobs were even more drastically impacted, as the nightly reservations limited the length of time a full-system job could run to 16 hours under ideal circumstances and considerably less than that in practice. This was exacerbated by the additional priority given to the CAPS jobs, which caused “near-miss” behavior where a `capability` job would have a reservation set due to being the highest priority until a CAPS job came along and caused the reservation to be lost. Because of this behavior, a `bigjob` QOS level was created and assigned to the `capability` queue. This new QOS has its own pool of reservations separate from the default reservation pool, so that the highest priority `capability` job will have a resource reservation even if there are other jobs with higher priority eligible to run. This impact on `capability` jobs also led to significant policy modifications at the center level, the most significant of which was that it was decided to place the 2010 CAPS Spring Experiment onto Athena rather than Kraken, as running a day or more worth of full-system jobs became a roughly weekly occurance on Kraken by early 2010.

3.3 User-Managed Scheduling on Athena

The first project given dedicated access to Athena was a climate modeling project from the Center for Ocean-Land-Atmosphere Studies (COLA) at the Institute of Global Environment and Society (IGES) [4]. This project used two major applications: IFS, an operational weather forecasting application from the European Center for Medium-range Wather Forecaster (ECMWF) [5]; and NICAM, a climate modelling application from the University of Tokyo which had previously been run only on the Earth Simulator [7]. Because Athena was dedicated solely to the COLA group, they requested that they be given ways to manage scheduling at a high level by tagging jobs which were more or less important. This was accomplished by giving the COLA users access to two new QOS levels, `bypass` and `bottomfeeder`. The `bypass` QOS applied the `NTR` ("next-to-run") policy to any jobs requesting it, allowing them to bypass the normal priority process. On the other hand, the `bottomfeeder` QOS disabled backfill and reservations for jobs that request it, causing them to run only when the system would otherwise sit idle.

The following table summarizes the queue times observed during the period of the COLA project's dedicated access to Athena from 1 Oct 2009 to 31 Mar 2010, during which the overall system utilization was 90.51%**:**

QOS	Jobs	CPU Hours	Min Queue Time	Max Queue Time	Mean Queue Time
`default`	11,851	37.4M	00:00:02	138:33:50	01:12:50
`negbal`	57	0.5M	00:00:02	06:19:08	00:37:19
`sizezero`	4,822	0.0M	00:00:01	148:22:35	01:31:55
`bypass`	1,540	32.2M	00:00:02	43:32:33	01:01:22
`bottomfeeder`	85	1.3M	00:00:03	137:09:04	22:27:44

Jobs using the `bypass` QOS saw slightly shorter queue times than those using the `default` and `sizezero` QOSes, which in turn saw significantly shorter queue times than jobs using the `bottomfeeder` QOS. The `negbal` QOS was only used by jobs over the course of two days due to an oversight with the COLA project's allocation on Athena, and its effect was minimal since the COLA project was the only group able to use the system. Overall, this arrangement was extremely successful, so much so that the COLA project produced significantly more data than they had originally anticipated.

One significant issue with this configuration was user confusion over the effects of the `bottomfeeder` QOS. After it was implemented, users complained that Moab was scheduling `bottomfeeder` jobs too aggressively. Upon more detailed inspection, it was found that some users who were running non-`bottomfeeder` jobs had large jobs that submitted `size=0` analysis or data transfer jobs when they ended, and those small jobs would in turn submit another large job. This would often result in situations where only `bottomfeeder` and `size=0` jobs were eligible to run, whereupon Moab could and would run both. After the affected users restructured their jobs to have large jobs submit their successor large jobs *before* the `size=0` job rather than *in* the `size=0` job, Moab gave the desired behavior for `bottomfeeder` jobs.

4. Conclusions

The use of QOS levels has become integral to scheduling on Kraken and Athena, particularly for projects which have scheduling needs outside the norm. They allow administrators to implement policy modifications and exceptions for individual users, groups, or accounts. Furthermore, they can also be used to enable users to manage the scheduling of their own workflows in a fairly straightforward, easy-to-use fashion.

References

[1] "CAPS", http://www.caps.ou.edu/.

[2] "Cluster resources :: Products - TORQUE Resource Manager", http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[3] "Cluster resources :: Products - Moab Workload Manager", http://www.clusterresources.com/pages/products/moab-cluster-suite/workload-manager.php.

[4] "COLA Home Page", http://www.iges.org/cola.

[5] "ECMWF Research", http://www.ecmwf.int/research/.

[6] "Job Flow", http://www.clusterresources.com/products/mwm/moabdocs/3.3jobflow.shtml.

[7] "NICAM Page", http://www.nicam.jp/hiki/.

[8] "Quality of Service (QoS) Facilities", http://www.clusterresources.com/products/mwm/docs/7.3qos.shtml.

[9] "The Weather Research and Forecasting Model", http://www.wrf-model.org/index.php.

[10] T. Baer and D. Maxwell, “Comparison of Scheduling Policies and Workloads on the NCCS and NICS XT4 Systems at Oak Ridge National Laboratory”, *Proceedings of the 2009 Cray User Group Meeting*, Atlanta, May 2009.

[11] M. Karo, et al. “The Application Level Placement Scheduler”, *Proceedings of the 2006 Cray User Group Meeting*, Lugano, Switzerland, May 2006.

About the Author

Troy Baer is an HPC systems administrator for the University of Tennessee's National Institute for Computational Sciences at Oak Ridge National Laboratory. He can be reached by emailing *<tbaer@utk.edu>*.

APPENDIX: `moab.cfg` Settings

```
### Default settings ###

# Priority settings
SERVICEWEIGHT    1
QUEUETIMEWEIGHT  1
RESWEIGHT        1
PROCWEIGHT       100
QOSWEIGHT        1000

# default QOS
QOSCFG[default]  PRIORITY=0

# negative balance QOS
QOSCFG[negbal]   PRIORITY=-100000

# size=0 QOS
QOSCFG[sizezero] QFLAGS=RUNNOW
QOSCFG[sizezero] PRIORITY=20000
QOSCFG[sizezero] QTWEIGHT=5000
QOSCFG[sizezero] QTTARGET=0:00:01
QOSCFG[sizezero] MAXPROC=0
QOSCFG[sizezero] MAXJOB=32

# job template to apply sizezero QOS
JOBCFG[sizezero.max]  TASKS=0

JOBCFG[sizezero.set]  QOS=sizezero

JOBMATCHCFG[sizezero] JMAX=sizezero.max
JOBMATCHCFG[sizezero] JSET=sizezero.set

# set default QOS list for all users
ACCOUNTCFG[DEFAULT] QDEF=default
ACCOUNTCFG[DEFAULT] QLIST=default,
negbal,sizezero
# NOTE:  The previous two lines should
# be on a single line.

# big job QOS
QOSCFG[bigjob] QFLAGS=RESERVEALWAYS,
DEDICATEDRESOURCE
# NOTE:  The previous two lines should
# be on a single line.

RESERVATIONDEPTH[bigjobs]   1
RESERVATIONQOSLIST[bigjobs] bigjob

# apply bigjob QOS to capability jobs
# by default
CLASSCFG[capability] QDEF=bigjob
CLASSCFG[capability] QLIST=bigjob,
default,negbal
# NOTE:  The previous two lines should
# be on a single line.
```

```
### CAPS forecasting additions ###

# CAPS forecast QOS
QOSCFG[capsforecast] PRIORITY=50000
QOSCFG[capsforecast] QTTARGET=0:05:00
QOSCFG[capsforecast] QTWEIGHT=1000
QOSCFG[capsforecast] QFLAGS=USERESERVED

# CAPS post-processing QOS
QOSCFG[capspostproc] PRIORITY=50000
QOSCFG[capspostproc] QTTARGET=0:15:00
QOSCFG[capspostproc] QTWEIGHT=1000
QOSCFG[capspostproc] QFLAGS=USERESERVED

# allow CAPS project to access
# additional QOSes
ACCOUNTCFG[TG-MCA95C006] QLIST=default,
sizezero,negbal,capsforecast,
capspostproc
# NOTE:  The previous three lines
# should be on a single line.

# standing reservations for forecasts
SRCFG[CAPSforecast-pm] PERIOD=DAY
SRCFG[CAPSforecast-pm] DAYS=Sun,
Mon,Tue,Wed,Thu
# NOTE:  The previous two lines should
# be on a single line.
SRCFG[CAPSforecast-pm] DEPTH=7
SRCFG[CAPSforecast-pm] STARTTIME=22:30:00
SRCFG[CAPSforecast-pm] ENDTIME=23:59:59
SRCFG[CAPSforecast-pm] TASKCOUNT=1205
SRCFG[CAPSforecast-pm] QOSLIST=capsforecast
SRCFG[CAPSforecast-pm] FLAGS=SPACEFLEX,
DEDICATEDRESOURCE
# NOTE:  The previous two lines should
# be on a single line.

SRCFG[CAPSforecast-am] PERIOD=DAY
SRCFG[CAPSforecast-am] DAYS=Mon,
Tue,Wed,Thu,Fri
# NOTE:  The previous two lines should
# be on a single line.
SRCFG[CAPSforecast-am] DEPTH=7
SRCFG[CAPSforecast-am]
STARTTIME=00:00:00
SRCFG[CAPSforecast-am] ENDTIME=06:30:00
SRCFG[CAPSforecast-am] TASKCOUNT=1205
SRCFG[CAPSforecast-am] QOSLIST=capsforecast
SRCFG[CAPSforecast-am] FLAGS=SPACEFLEX,
DEDICATEDRESOURCE
# NOTE:  The previous two lines should
# be on a single line.
# standing reservations for post-
# processing
SRCFG[CAPSpostproc-pm] PERIOD=DAY
SRCFG[CAPSpostproc-pm] DAYS=Sun,
Mon,Tue,Wed,Thu
# NOTE:  The previous two lines should
# be on a single line.
SRCFG[CAPSpostproc-pm] DEPTH=7
SRCFG[CAPSpostproc-pm] STARTTIME=22:45:00
SRCFG[CAPSpostproc-pm] ENDTIME=23:59:59
SRCFG[CAPSpostproc-pm] TASKCOUNT=85
SRCFG[CAPSpostproc-pm] QOSLIST=capspostproc
SRCFG[CAPSpostproc-pm] FLAGS=SPACEFLEX,
DEDICATEDRESOURCE
# NOTE:  The previous two lines should
# be on a single line.

SRCFG[CAPSpostproc-am] PERIOD=DAY
SRCFG[CAPSpostproc-am] DAYS=Mon,
Tue,Wed,Thu,Fri
# NOTE:  The previous two lines should
# be on a single line.
SRCFG[CAPSpostproc-am] DEPTH=7
SRCFG[CAPSpostproc-am] STARTTIME=00:00:00
SRCFG[CAPSpostproc-am] ENDTIME=06:45:00
SRCFG[CAPSpostproc-am] TASKCOUNT=85
SRCFG[CAPSpostproc-am] QOSLIST=capspostproc
SRCFG[CAPSpostproc-am] FLAGS=SPACEFLEX,
DEDICATEDRESOURCE
# NOTE:  The previous two lines should
# be on a single line.


### COLA dedicated additions ###

# bypass QOS
QOSCFG[bypass]       QFLAGS=NTR

# bottomfeeder QOS
QOSCFG[bottomfeeder] PRIORITY=-100000
QOSCFG[bottomfeeder] QFLAGS=NOBF,
NORESERVATION
# NOTE:  The previous two lines should
# be on a single line.

# allow COLA project to access
# additional QOSes
ACCOUNTCFG[UT-NTNL0021] QDEF=default
ACCOUNTCFG[UT-NTNL0021] QLIST=default,
bypass,bottomfeeder,negbal,sizezero
# NOTE:  The previous two lines should
# be on a single line.
```