# Blue Waters Resource Management and Job Scheduling Best Practices

Sharif Islam, Brett Bode,
Jeremy Enos
National Center For Supercomputing Applications
University of Illinois, Urbana IL
mislam,brett,jenos@illinois.edu

## ABSTRACT

This paper describes resource management and job scheduling best practices learned from operating Blue Waters (a petascale Cray XE+XK supercomputer with 26,864 compute nodes) since April 2013. We will describe various aspects of such operation while focusing on the challenges experienced while maintaining a large, shared computational resource such as Blue Waters.

## CCS Concepts

• Social and professional topics→Professional topics→Management of computing and information systems.

## Keywords

Job scheduling, torque, moab, scheduler best practices.

## 1. INTRODUCTION

This paper describes resource management and job scheduling best practices learned from operating Blue Waters [1],[3] (a petascale Cray XE+XK supercomputer with 26,864 compute nodes) since April 2013. We will describe various aspects of such operation while focusing on the challenges experienced while maintaining a large, shared computational resource such as Blue Waters that routinely handles three to five thousand queued and a couple of thousand running jobs of varying sizes (from full system to single node workloads). The biggest challenge for the scheduler is managing the turnover of large jobs which range from 1,000-20,000 nodes while attempting to minimize the drain cost. While designing and implementing our policies and best practices we kept in mind the diverse and competing user needs along with our commitment to minimum interruptions to the community. The National Science Foundation has awarded a very wide range of science disciples and Blue Waters is committed to support these diverse workloads. While many of those teams are experienced users of HPC resources others have had a much larger learning curve and many are running at very large scale for the first time. At the same time, we also support and indeed have driven the need for the latest features and enhancements available in Torque and Moab with a rigorous regime of test suites and aggressive upgrade schemes.

## 2. RESOURCE MANAGER AND SCHEDULER SUMMARY

Blue Waters uses Torque as the resource manager and Moab as the scheduler within the Cray environment. We also enabled the Topology Aware Scheduler feature for the Cray 3D Torus in Moab in January 2015[4]. Topology aware scheduling requires jobs to be placed on sets of nodes contiguous on the torus network and be of a certain shape based on the node count (such as 8x2x8 or 11x8x24 cuboids). Significant testing demonstrated that the overall science throughput of the system was higher using the topology aware mode despite the lower overall node occupancy. However, this mode also introduces sets of challenges and performance issues that are not present in the regular Moab version and requires a different set of best practices, which is not in the scope of this paper. For example, based on the nodes requested by the user, Moab searches the whole system and finds the best placement that matches the shape associated with that particular node count (the shapes are pre-defined inside Moab). For some of our node ranges we created job templates to override the default shape table to ensure better job placement. Enabling this mode also impacts scheduler performance, utilization, and iteration length. We are working with Adaptive Computing to increase the efficiency of the scheduler.

In Blue Waters, Torque and Moab manage 26,864 compute and 64 mom nodes with a default feature of "xe" for Cray XE nodes and "xk" for Cray XK (GPU) nodes. These and other node features are managed via torque node files. The master Moab configuration file moab.cfg resides in the Moab home directory (/var/spool/moab/etc). We take advantage of the #INCLUDE feature in Moab to organize our configuration files. Instead of having everything in one file we separate out the various options that might require more frequent changes than the global options. For example, the setting RESERVATIONDEPTH is in moab.cfg but we have another reservation depth based on a specific QoS which is included in the include.qos file and called from the moab.cfg file via the "#INCLUDE include.qos" line. Any significant changes in the configuration are vetted through a change control process (certain day to day operational items are exempt from that). We also keep a repository of old configuration files and detailed change log for each change. All the changes go through a set of tests and simulations (see section 10).

## 3. SCHEDULING POLICIES

This section will briefly highlight some of our scheduler policies and best practice principles behind them.

### 3.1 Job sizes, Queues, and Priorities

In Blue Waters, larger jobs generally get priority over smaller jobs. However as a best practice we make sure that smaller jobs are not waiting in the queue for a long time. This is achieved in three different ways. First, we have discounts available for jobs that backfill, jobs with accurate wall clock time, and jobs using flexible wall clock time (see section 3.3). This encourages users to take advantage of available and varying backfill windows as the system drains for larger workload and, as a result, shortens the queue wait time while also benefiting system utilization and scheduler effectiveness. Second, we have priority-based queues (normal, high, low, debug) that incur different charge factors. Depending on user needs we suggest the use of "high" or "debug" queue instead of "normal" or "low" queue for quicker start time as those queues are guaranteed to get a higher priority and priority reservation. Third, our priority factors use different weights such as job sizes, wall time, queue time and expansion factor to ensure that different aspects of the job are getting weighed in priority calculations. We have also spent considerable time in understanding "drain time" incurred by large jobs [5][1] and came up with a method to calculate this time. This method helps us understand how the system is behaving while running and waiting for varying workloads.

The challenge still remains to support workloads of varying sizes and length. As a best practice we strive to provide as much information as possible to the user community regarding our scheduler policies and settings without going into the gory details of the Moab configuration. This helps to set user expectations. We also routinely monitor queue status (section 7) and policy (section 8) to understand and improve scheduler behavior.

### 3.2 Usage Throttling

Ideally we would like to schedule and run as many jobs as possible. However, several recent bugs in Moab and Torque introduced performance issues such as long iteration times and Torque-Moab communication breakdowns when there were increased numbers (>2000) of jobs, user and standing reservations in place. This is partly due to the large numbers of nodes Torque and Moab have to handle for Blue Waters. As mentioned earlier, topology aware mode also makes this issue slightly more complicated. Adaptive Computing is actively working on some of these bugs and other enhancements in collaboration with Cray and NCSA. In the meantime, we have put a few usage limits in Moab to address the present scheduler issues.

Due to the large number of nodes and jobs in Blue Waters we frequently notice long scheduler iteration times (from 300-600 seconds). Applying usage throttle options helps to maintain the iteration time to a manageable level for interactive jobs and job turn around. We use MAXIJOB to limit the total number of jobs that can be eligible at any given time. Besides maintaining a reasonable iteration time MAXIJOB also helps in preventing large numbers of jobs accruing priority all at once and strives to improve fairness across multiple users and projects. At the moment non-eligible jobs do not accrue priority.

### 3.3 Preemption and Flexible Wall Clock Time

Flexible wall clock is used via the "–l minwclmit" flag in qsub. This option can improve job turn-around time and utilization. By providing a minimum wall clock time for a job to start as well as a maximum time, the scheduler can start a job early and try to keep the job running by extending the working wall clock time in increments of 30 minutes for as long as the nodes remain available. Once the minimum wall clock time is reached, the job will become preemptible by higher priority jobs. We recommend use of the flexible wall clock option (minwclimit) for workloads that can make progress with a portion of the total requested wall time, as it usually results in an earlier start time, distinguishing it from use of the preemptee flag by itself.

### 3.4 Fair share

Our initial setup did not include fair share and this was intended to reflect our scheduler policy that supports quick turnaround for large jobs. However, after a year of operation we started noticing certain teams were waiting longer than others for their jobs to run. This is partly due to lack of fair share and also due to varying allocations that were granted. Teams with larger allocation also tend to have more resources to debug and test their codes and run large numbers of jobs more frequently. After evaluating several months of workloads and testing we decided to introduce group level fair share to ensure that a single team or user cannot dominate the queue. Our FSINTERVAL currently set to the maximum allowable walltime and FSDEPTH is seven days. For the test, we copied the fair share data files (/var/spool/moab/stat/) to the Moab monitor mode setup and observed the changes via mdiag –p.

### 3.5 Job Dependencies

We have users that take advantage of Torque's job dependency feature to chain several jobs together. As more and more people started chaining jobs we noticed similar size jobs not getting scheduled and running back to back. As a result certain job sizes waited longer in the queue. The reason was dependent jobs were only accruing priority when they became eligible after the parent job finished. To remedy this we added priority exception settings (via jobprioexceptions and jobprioraccrualpolicy) to ensure all the jobs that are held due to dependency are accruing priority. After this change similar sizes jobs that were held due to dependency had quicker turnaround.

But this accrual policy caused an unintended outcome. When the held jobs become eligible due to the accrual of priority it may receive a priority reservation right away and as a result disrupt the current reservation order so we decided to remove this feature. This is particularly problematic when the system has already drained for a large job which gets bumped by a newly eligible

[1] We define 'Drain Time' as: "for a single node is the amount of time that node is held in reserve and prohibited from running workload in order to allow enough nodes to become available to start another job. Since jobs run across multiple nodes, the metric, more loosely referred to as 'Drain Time', is actually the sum of time spent by a set of nodes and is measured in node hours(nHrs), or node-seconds(nSecs). This is in line with how jobs are measured in their utilization of a system and the capital in which an allocation is granted."

member of a job chain resulting in repeated large drain costs. As a workaround we accommodate special requests by creating reservations (section 9) to guarantee quicker job turn around for chained workloads.

## 4. SUBMIT FILTER

We use the torque submit filter to check user inputs upon job submission. This allows us to check various misconfigured and incorrect options users may include in the job script. We have the flexibility to alert the user or reject the job. Ideally Torque and Moab should do some of these sanity checks but this way we can implement some of our own customized checks. We make sure to provide adequate explanation for rejecting the job to prevent further queries and tickets from the user. The filter also checks user allocation by querying a database server and rejects the job if allocation has expired or exhausted.

## 5. RESERVATION DEPTH

Currently we use a global reservation depth of 10. Initially we opted for a higher reservation depth in order to provide the users with a better estimate of job start time via showres as without a reservation showstart provides both inaccurate information and can DOS the Moab server. In addition, a reservation depth of 10 allows the scheduler to better block out future job placement providing a better turnaround for jobs that are not easily backfilled, but that still can be placed around the reservation for the top priority job. However, increasing the reservation depth caused various scheduler efficiency issues. We also ran into issues with two different resources in the system: XE (22,636 nodes) and XK (4228 nodes) where the top priority reservation often times ended up being the XE workload. To fix the issue we created a QoS for XK workload and assigned a reservation depth to that QoS.

## 6. JOB TEMPLATES

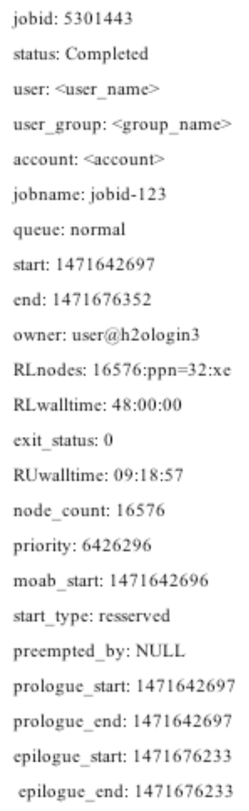
We make use of job templates to alter default Moab behavior and ensure better job placement as it makes a significant impact for the Topology Aware Scheduler. We have templates for jobs that are long-running but with small node count. Based on torque node features we place these jobs to a specific region of the torus network, to preserve large contiguous blocks of nodes for larger jobs.

## 7. MONITORING AND REPORT

We have various monitors and alerts in place in order to ensure service availability and general health of the Torque and Moab servers. For example, we have test harness in place via Jenkins that runs various user commands (such as qstat, showq) to check uptime and service daemon availability. If trqauthd daemon dies in the torque server qstat will fail and Jenkins will alert us about this. We also have alerts in place for various ERROR messages in the moab log. Iteration times are monitored and admins are also alerted when a certain threshold is reached. Long iteration times are normal in Blue Waters but excessive iteration times prompt us to take a closer look at Moab logs and the running jobs.

### 7.1 Job Statistics

We collect a variety of data for each job and store them in a Mysql database (see Figure 1). These are collected both from torque and moab logs. These job data help us query large numbers of job status and create various reports that are regularly presented to the funding agencies and the user community.



```
jobid: 5301443
status: Completed
user: <user_name>
user_group: <group_name>
account: <account>
jobname: jobid-123
queue: normal
start: 1471642697
end: 1471676352
owner: user@h2ologin3
RLnodes: 16576:ppn=32:xe
RLwalltime: 48:00:00
exit_status: 0
RUwalltime: 09:18:57
node_count: 16576
priority: 6426296
moab_start: 1471642696
start_type: resserved
preempted_by: NULL
prologue_start: 1471642697
prologue_end: 1471642697
epilogue_start: 1471676233
 epilogue_end: 1471676233
```

**Figure 1: Blue Waters job statistics.**

## 8. ACCOUNTING

We currently use torque job data to charge node hours. However, we are actively testing Moab Native Accounting Manager (NAM) to gather better job metadata and reservation information that is not provided by the torque accounting data. Also, we had several major bugs with Torque accounting after a recent upgrade that resulted in significant efforts in our part to ensure the accuracy of job usage data. Even though the major bugs in Torque have been addressed our recent tests indicate accounting data received from Moab to be more reliable and accurate along with including the ability to provide a charge for unused time in a reservation.

The torque server runs a cron job that sends job data to our external database server which in turn calculates the job charges and other discount information. Our best practices for accounting ensure timely availability of usage data to the users via the portal and the command line interface. We also have mechanisms in place for error detection and deploy temporary work around in the event of bugs introduced due to Torque upgrade. We also use Torque data to populate XDMoD[7] which has capability to provide a wide range of metrics for jobs running on Blue Waters.

## 9. SPECIAL USER RESERVATIONS

Often times we receive special requests from users for various types of reservations. We try our best to accommodate these requests without interrupting other users. For example, recently we created two 1024-nodes reservations for a user for several days in order to meet a deadline. While the reservations were running we had alerts in place to make sure there were jobs submitted to the reservation (via advres flag in qsubs). This was put in place to ensure that the reserved nodes are actually getting utilized and not sitting idle.

## 10. TEST AND SIMULATION

Moab provides a simulation mode where actual workload via event trace files can be loaded. However, this simulation capability is currently limited and contains several bugs so we are not using this feature. Instead we designed a test and simulation environment using our test system and Native Resource Manager (NativeRM).

Our test system is used to verify and run new software. Blue Waters always attempts to run the latest Torque and Moab software in order to take advantage of the bug fixes and various enhancements. Before deployment we first install the latest bits in our test machine. Admins and application support staff run test jobs to ensure basic functionalities. We also go through the latest bug fixes to verify the changes and improvements. Based on the test results and absence of any new bugs and regressions we submit a change control and schedule a deployment. Usually, we update both Torque and Moab while regular workloads are running without any issues.

NativeRM and Moab are used to verify scheduler policy changes to make sure certain jobs will run after the change. NativeRM cannot handle event traces but a job submission script can be generated based on such traces in order to be used in the NativeRM set up to verify scheduler policies.

## 11. PERIODIC REVIEW OF SCHEDULER POLICY

We have a bi-weekly meeting where we revisit various scheduler policies and issues reported by the users (this is beyond our regular admin status and bug meetings). Not just system administrators who are in charge of the scheduler but also application support and storage admins attend this meeting. This meeting provides a venue to look at how scheduler policies are impacting the whole system. Various recommendations are approved here and put forth for tests and deployment.

## 12. BUG REVIEW

We have weekly bug review meetings with Adaptive Computing where we discuss bug progress and work around. Due to large numbers of bugs and RFEs submitted for Blue Waters, a weekly meeting helps us to keep track of the issues and plan for future deployment. This regular meeting helps in forging a strong relationship with the vendor. To prepare for the meetings we make sure that we understand the issue at hand properly and are able to communicate succinctly with the help of various log data. At the same time we also prioritize bugs and RFEs in the following categories: urgent, critical, major, minor and categorize them based on issues such as accounting, preemption. Having a updated and organized list of bugs that can be sorted and filtered facilitates the bug review discussions.

## 13. DOCUMENTATION AND USER COMMUNICATION

We maintain two sets of documentation that are part of the daily operations. User documentation is accessible online in the portal which provides hardware summary, programming environment and compiler information along with various pages explaining job scheduler policies. We encourage users to use showbf to gather up-to-date backfill information. The showbf data is also visually represented in the portal[6].

Providing the necessary documentation and effective user communication are essential for successful operation. We also have internal documentation that provides various configuration and troubleshooting details for the admin team. These are in addition to the vendor documentation. We regularly update both sets to ensure the documentation reflects the current status of the system.

## 14. CONCLUSIONS

We are continuously engaged with our vendors and user community to provide a better job-scheduling environment. Managing the resource manager and scheduler in a large system like Blue Waters bring in unique challenges for performance and efficiency. Here we presented our approach to addressing these challenges and best practices that help us utilize the system in an efficient manner. Some of these practices can be applied to any system and other products as well (not just Torque and Moab).

## 15. ACKNOWLEDGMENTS

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

## 16. REFERENCES

[1] Blue Waters: Sustained Petascale Computing https://bluewaters.ncsa.illinois.edu/. Accessed: 2016-08-22.

[2] Blue Waters User Documentation https://bluewaters.ncsa.illinois.edu/documentation/. Accessed: 2016-08-24.

[3] Brett B. et al. 2012. The Blue Waters Super-System for Super-Science. Contemporary HPC Architectures. *The Blue Waters Super-System for Super-Science*. Contemporary HPC Architectures, Jeffery Vetter ed. Sitka Publications. 339-366.

[4] Enos, J. et al. 2014. Topology Aware Job Scheduling Strategies for Torus Networks. Proceedings of the Cray User Group meeting (Lugano, 2014).

[5] Fullop, J. The Hidden Cost of Large Jobs Drain Time Analysis at Scale. Proceedings of the Cray User Group meeting (London, 2016).

[6] Why isn't my job running?

https://bluewaters.ncsa.illinois.edu/why-isn-t-my-job-running: Accessed: 2016-08-24.

[7] XSEDE Metrics on Demand (XDMoD) http://xdmod.ncsa.illinois.edu/. Accessed: 2016-08-22.