Impact of Reservations on Production Job Scheduling

Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, Phil Andrews
San Diego Supercomputer Center, University of California, San Diego
{mmargo, kenneth, pkovatch, andrews} @sdsc.edu

Abstract

The TeraGrid is a closely linked community of diverse resources: computational, data, and experimental, e.g., the imminent very large computational system at the University of Texas, the extensive data facilities at SDSC, and the physics experiments at ORNL. As research efforts become more extensive in scope, the co-scheduling of multiple resources becomes an essential part of scientific progress. This can be at odds with the traditional management of the computational systems, where utilization, queue wait times, and expansion factors are considered paramount and anything that affects their performance is considered with suspicion. The only way to assuage concerns is with intensive investigation of the likely effects of allowing advance reservations on these performance metrics.
To understand the impact, we developed a simulator that reads our actual production job log and reservation request data to investigate different scheduling scenarios. We explored the effect of reservations and policies using job log data from two different months within consecutive years and present our initial results. Results from the simulations suggest that utilization, expansion factor and queue wait time indeed can be affected negatively by significant numbers and size of reservations, but this effect can be mitigated with appropriate policies.

1. Introduction

Scientists need to reserve computational resources for different reasons. For instance, a scientist may need resources at a specific time to meet a deadline or perform a demonstration. A scientist may also want to co-schedule multiple resources. Creating reservations is one way of allowing jobs to run simultaneously at more than one site or resource and guarantees that a resource is available at a specific time. Many local schedulers can implement policies and operate independently from a centralized scheduler without requiring coordination between sites with different scheduling goals. This is in contrast with other co-scheduling mechanisms [1,2,3].

Reservations may reduce batch job throughput of a system, as measured by utilization, average expansion factor and average queue wait time. A study of the effect of reservations on utilization and expansion factor [4] concluded that meta-scheduled reservations equivalent to up to 15 percent of the workload result in less impact than fixed time slots set aside for cross-site jobs. In that study, a real workload was used to compare baseline utilization and expansion factor in the presence and absence of synthetic reservations. In this study, we sample real user reservations to find the point at which user reservations interfere unacceptably with the batch workload. We seek to determine whether policies restricting reservation creation can moderate the negative effect of reservations on utilization, expansion factor and queue wait time. In some cases, the effect of reservations can even be advantageous: allowing a "clean start' with an opening for large, efficient jobs to begin.

The San Diego Supercomputer Center [5] offers several compute resources including DataStar. DataStar consists of 2,528 Power4+ processors connected via IBM's Federation High Performance Switch (HPS). 272 of the nodes have eight processors per node and 11 of the nodes have 32 processors. In aggregate, the

machine has over 7 TB of memory. This machine has been in production since April of 2004 and delivers over 14 million service units (SU, representing one hour of processor wall-clock time) per year. DataStar runs LoadLeveler [6] as the resource manager and Catalina [7] as the scheduler. DataStar is also part of the TeraGrid.

The TeraGrid [8] is a multi-year effort sponsored by the National Science Foundation (NSF) to construct the nation's largest grid. The TeraGrid hosts a variety of diverse resources including over 100 TF of High Performance Computing (HPC) power and a spallation neutron resource: an accelerator-based neutron source [9]. These resources are connected via a dedicated high-speed network of 10-30 Gb/s. User allocations are fungible at different sites throughout the TeraGrid. A General Parallel File System (GPFS) [10] is exported from SDSC enabling users to access the same files at different sites without copying files around. This capability facilitates metascheduling.

The TeraGrid currently consists of nine sites: the National Center Supercomputing Applications, San Diego Supercomputer Center (SDSC), Argonne National Laboratory, the National Center for Atmospheric Research, Pittsburgh Supercomputer Center, Texas Advanced Computing Center, Indiana University, Purdue and Oak Ridge National Laboratory.

The TeraGrid is interested in metascheduling capabilities including user-settable reservations [11]. User-settable reservations are favored since they are more efficient with staff time. A Metascheduling Requirements Analysis Team (RAT) [12] was formed to explore the capabilities, Resource Provider (RP) requirements, and software technologies required to assist in the formal adoption of metascheduling capabilities. RPs have expressed concerns about how their local scheduling goals will be affected by metascheduling capabilities, including reservations. Most sites are planning to implement reservations but are unsure how reservations will impact their queues and overall machine utilization. They are considering how to set policies that would help ameliorate the effect.

Using a simulator with real job log data from the actual production queue on DataStar, we can see how local jobs are affected by the reservations from the co-scheduled jobs and compare the results without the reservations. The simulator gives us a controlled and repeatable environment to see the outcomes of the scheduling decisions made by the policies of the scheduler. With this we can assess the impact of reservations and make appropriate policies. For instance, a site may want to limit the impact of reservations created by a specific user. This site could create a policy within the scheduling software to only allow a limited number of reservations created by that user. Some sites want to offer the co-scheduling feature with reservations and are willing to balance any potential decrease in their overall utilization by introducing increased charges for reservations.

The evaluation of parallel job scheduling algorithms is fraught with problems, a long but probably not exhaustive list forms the "Frachtenberg pitfalls"[13]. It is beyond the scope of this paper to demonstrate the avoidance of all 32 of these, but the great majority are circumvented by the technique of using a real workload, rather than a simulated one. The jobs are actual submissions to the SDSC machine DataStar, with the addition of putative reservations to determine the impact on the gross queuing parameters such as usage and queue waits, with and without throttling restrictions.

2. Hypothesis

While user-settable reservations would enable co-allocated grid jobs and make efficient use of staff time, system administrators are often reluctant to allow users to make reservations at

will. This stems from a fear that injecting reservations will disrupt the job queue, resulting in degraded utilization, average expansion factor and average queue wait time. We hypothesize that the impact of user-settable reservations on the job queue can be controlled through policies throttling reservation creation. The scheduler used at SDSC and in this simulation study allows users to create reservations only after all eligible jobs have been given reservations. This means that the existing job queue is not affected by user-settable reservations. Subsequent jobs would need to be scheduled around those reservations.

The policy we use to throttle user-settable reservations is that the total node-seconds of user-settable reservations within any sliding 7 day period must not exceed 256 nodes * 12 hours worth of node-seconds. This represents all unshared batch nodes in the system for half a day each week.

3. The Simulator

We developed a simulator to explore the impact of reservations on a real production workload. The simulator allows us to replay the exact workload, vary the number of reservations and observe the effects. We selected three parameters to evaluate the effects of reservations: machine utilization, expansion factor and queue wait time.

Diagram 1 depicts the five major components of our simulator: Simulation Core, Catalina, LoadLeveler simulators, and accounting and reservation databases. The core is written in Python and uses SimPy simulation framework [14]. The simulation core depends heavily on the simulation clock. Our simulator is designated DES (Discrete Event Simulator). In DES, time can be advanced into a discrete time where an event is happening; other times of non-activity are simply skipped. The core is responsible for keeping track of active jobs, queued jobs, and exiting jobs, adjusting the discrete simulation clock, and keeping track of current and future reservations.

The next component is Catalina [7]. Catalina is our production scheduler on SDSC's Datastar. It is written in Python. Catalina is stateless. A scheduling iteration is independent than that of the past and that of the future.

LoadLeveler [6] is Datastar's resource manager software from IBM. It is responsible for accepting job submissions and to signal the nodes to run jobs. It has many different components and a set of APIs to access its functions. To become an external scheduler, Catalina uses many of the LoadLeveler API calls to perform its duty. For example, it uses API calls to query the status of jobs, machines, resources, adapters, etc. Catalina also uses the API to instruct LoadLeveler to run a job on a node. Since we are going to run a discrete event simulator, LoadLeveler behavior must be simulated. LoadLeveler is not stateless; therefore it is impossible for the simulator to advance the simulation clock discretely. To overcome this problem, we wrote LoadLeveler wrapper scripts.

The LoadLeveler API calls that we need are qj_ll, llq, and rj_ll. qj_ll or query job LoadLeveler queries LoadLeveler for all jobs known to LoadLeveler. qj_ll outputs the name of the job, owner, group, node amount requested, wall clock time requested, switch adapter window requirement, along with other job properties. When a job is queued, its information will be included in qj_ll output.

llq or LoadLeveler query works similarly to qj_ll. llq records current job information and display them in human readable format.

rj_ll or run job LoadLeveler is a wrapper script written in C to start a job currently in the queue.

SDSC has accounting and reservation databases. These databases are updated regularly. The accounting database consists of finished jobs information; such as job id, owner, group, account, charges, amount of wall clock time used, return code, location to

binaries, location to input and output files, etc. For historical purposes, we can query the database and obtain information about jobs that were submitted by a particular user on a particular day in the past.

The reservation database stores information about user settable reservations on Datastar. It has information about the reservation creation date, reservation start date, reservation end date, owner, etc.

The simulation consists of cycles. A cycle is broken down into five steps: the input step (Step 1), the recording step (Step 2), the scheduling step (Step 3), the execution step (Step 4), and the reporting step (Step 5). The simulation terminates if all jobs and reservations in accounting and reservation databases have been completed or the time limit has been reached. The time limit should be set to be the time when all the jobs and reservations finishes plus few days of buffer period.

In the beginning of the simulation, we initialized the simulation clock to zero, read the job accounting and reservation database, and load them to memory in a queue sorted by their relative submit time. Each job and reservation has a relative submit time. For example, a job submitted on midnight, January 2, 2007 has relative start time of 86400 seconds, if the simulation begins on January 1, 2007. This technique allows us to distribute jobs according to real life distribution. Reservations are distributed uniformly across simulation time.

In Step 1, the simulation core reads the value of simulation clock and reads the memory for any job that submits at that time. If such job or reservation exists, it will update the next simulation time to the next job submission time or reservation creation time in memory and dequeue the current job or reservation.

In Step 2, any jobs extracted from Step 1 are inserted to the qj_ll and llq simulator. Effectively, the jobs information are registered to LoadLeveler. This is the recording step. Also, in this step, reservations extracted from Step 1 are submitted to Catalina. If there are no jobs or reservations extracted from Step 1, the simulation continues to Step 3.

In Step 3, the simulator calls Catalina to run a scheduling iteration. In this iteration, Catalina reads qj_ll and llq that are already populated with jobs in Step 2. Therefore, the newly submitted jobs are now being considered for execution. Also in this step, reservation requests are either granted or rejected depending on the active reservation policy. In Step 3, Catalina passes log information about active jobs and reservations to the core. Look at the text below for more detail. To reduce the amount of processing time, the scheduler iteration occurs every fifteen minutes of simulation time between job submission or completion.

```
 43258  :: job (ds002.236406.0) requesting
(6) node is here
 43639  :: job (ds002.236408.0) requesting
(6) node is here
 44100  :: ********** begin executing
catalina_schedule_jobs.py *********
 44100  :: ********** executed
catalina_schedule_jobs.py ********
 44110  :: job(ds002.236408.0) has been
waiting for (471) and now it is running.
 44110  :: total node usage is now (53) or
(20.00) percent
 44110  :: job(ds002.236406.0) has been
waiting for (852) and now it is running.
 44110  :: total node usage is now (59) or
(22.00) percent
 44321  :: job(ds002.236408.0) is done
running. it ran for (211.0) seconds
 44321  :: total node usage is now (53) or
(20.00) percent
 44334  :: job(ds002.236355.0) is done
running. it ran for (31724.0) seconds
 44334  :: total node usage is now (51) or
(19.00) percent
 44795  :: job(ds002.236406.0) is done
running. it ran for (685.0) seconds
 44795  :: total node usage is now (45) or
(17.00) percent
 45000  :: ********** begin executing
catalina_schedule_jobs.py *********
 45000  :: ********** executed
catalina_schedule_jobs.py ********
```

In Step 4, Catalina checks its internal tables to

see if it can run one or more jobs at present. If so, it calls rj_ll simulator with the eligible job ids as input parameters. It logs this action to simulation core. At the end of this executing step, the simulation cleans finished jobs by flushing their entries from llq and qj_ll simulator internal table. If there is no job to be run at present, Catalina skips this step. Reservations are handled analogously.

In Step 5 or the reporting step, the simulator print logs into standard output. We can see a message with a timestamp for each action performed on the job or reservation. The text above is an example of our simulator output.

At the end of Step 5, simulation clock is advanced to the next event time and Step 1 restarts. Output from Step 5 allows us to calculate wait time for each job, expansion factor for each job, and instantaneous overall utilization at various point in time. With this information, we can calculate higher-level metrics presented in later sections.

The simulation strives to provide the actual production environment with jobs arriving to the queue at the same distribution as the historical job distribution. The jobs we simulate are a copy of actual jobs from Datastar. We also simulate jobs that finished earlier than its requested wall clock time. The simulator uses historical reservation information from TeraGrid. In the base case, we can see how our simulator resembles the historical production schedule. One factor that deviates us from actual production schedule is the fact that we don't simulate node failures, either hardware or software.

We wrote ~1000 lines of code to implement the simulation core and the LoadLeveler simulator components. The code for the Simulator will be freely available after approval by UCSD's Technology Transfer Office. It will work with generic queue-based external schedulers and resource managers.

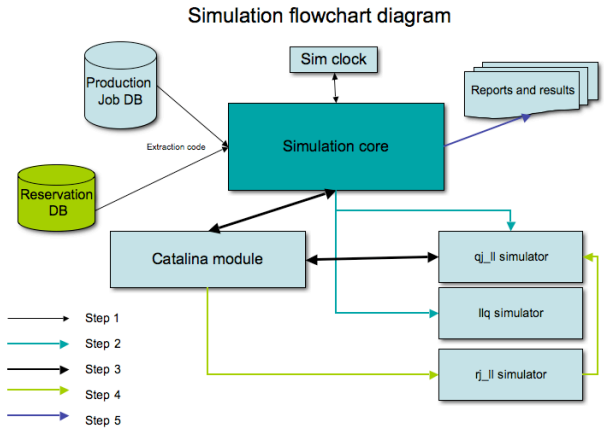


Diagram 1. Simulation Flowchart.

4. Evaluation Criteria

To judge the impact on our scheduling goals, we examined the average queue wait time for each job, the expansion factor for each job, and total utilization of the machine. Queue wait time measures the length of time between when a job comes into the queue and when it starts running. It is measured in hours. The expansion factor is the ratio of the sum of requested wall-clock time and queue wait time divided by the requested wall-clock time. Expansion factor is unit-less; the lower the expansion factor is, the better. Finally, utilization is the percentage of the entire cluster being used during the simulation.

To measure impact on the job queue, we use utilization, average expansion factor and average system queue wait time:

$$Utilization = \frac{\sum job\_node\_seconds}{Available\_node\_seconds}$$

Equation 1. Utilization equation

$$Average\_expansion\_factor = \frac{\sum_{alljobs}(exp\_factor * job\_node\_seconds)}{\sum_{alljobs} job\_node\_seconds}$$

Equation 2. Average expansion factor equation

$$Average\_system\_queue\_wait\_time = \frac{\sum_{all\_jobs}(job\_wait\_time * job\_node\_seconds)}{\sum_{all\_jobs} job\_node\_seconds}$$

Equation 3. Average system queue wait time equation

5. The Simulation

We wanted to see the effect of real, requested reservations on our normal workload. We also wanted to see how many reservations it would take to significantly interfere with our scheduling goals. Lastly, we wanted to know if our scheduling goals could be maintained by using policies.

To accomplish these objectives, we planned three runs: (1) a baseline run using only the real production workload, (2) a run including real reservation requests and (3) a run with policies enabled to control the number of reservations. To really show the effect of reservation in our production job mix, we used one year worth of reservations on our system over a month period of time.

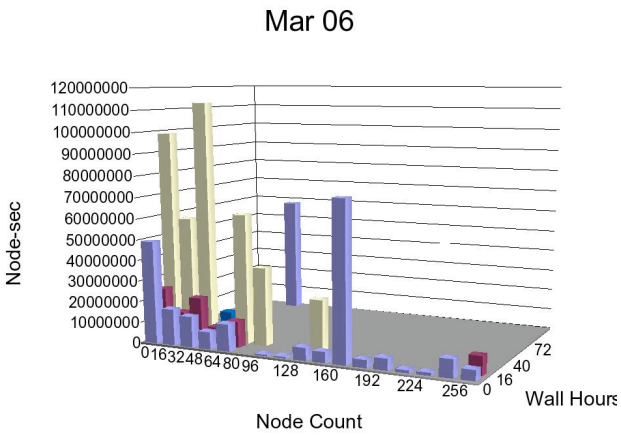
Additionally we use two different job mix, one from March 2006 and one from March 2007. The reservation data however, are the same between the two sets of runs.

For the simulation, we used DataStar's real production workload from the two months, based on job submission time.

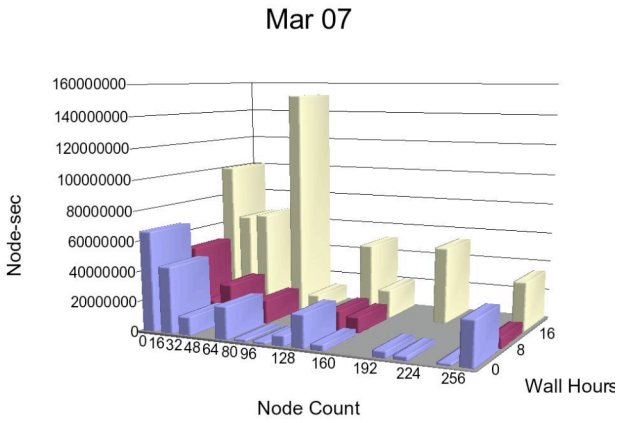
In the first week of simulation time, there were 767 jobs in the queue. The job mix included jobs from one to 256 nodes (full machine) with a duration of one to 18 hours (the maximum time that can be requested). Six jobs requested the full machine and 29 requested 128 nodes or more. We injected 15 reservations during each week of simulation time,, all actually requested through TeraGrid, with job sizes from eight to 128 with a duration of four to 18 hours.

In 2006, TeraGrid users requested reservations for 60 co-scheduled jobs. We used these real-life requests as the basis for our reservations for both simulations. Compressing a year's worth of actual reservations into a month of simulated job scheduling should overstate the impact of real reservations on the system.

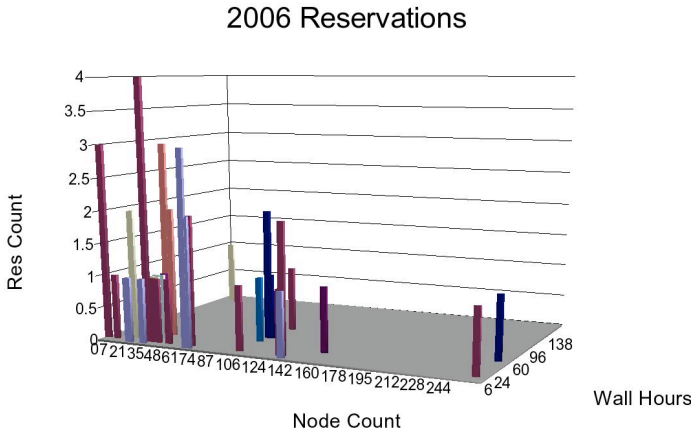
Graphs 1 and 2 show the number of jobs requested at each node count for 2006 and 2007. The colors are not significant, only to help identify the jobs. Graph 3 shows the reservations requested for 2006, which was used for both the 2006 and 2007 simulations.



Graph 1. Job Histogram for March 2006.

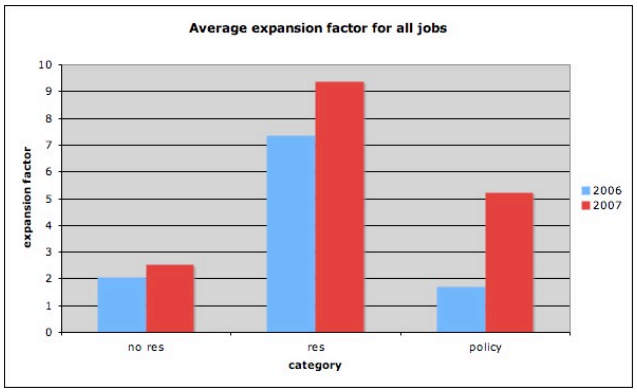


Graph 2. Job Histogram for March 2007.

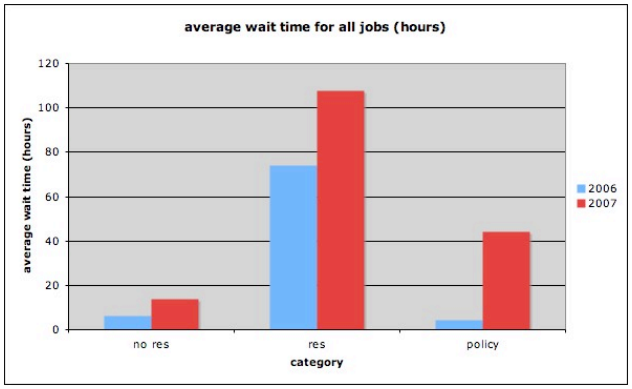


Graph 3. Reservation Histogram for 2006.

6. Results



Graph 4. Average Expansion Factor for All Jobs



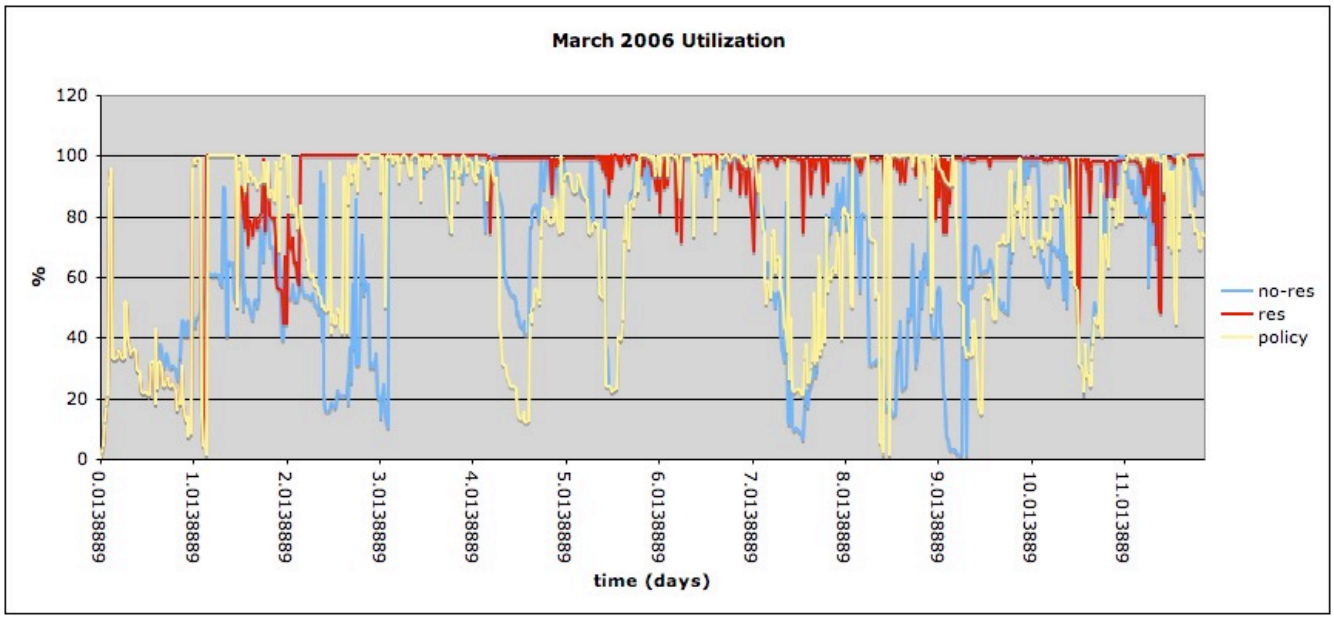
Graph 5. Average Wait Time for All Jobs

From the job data from March 2006, 722 jobs completed during the simulation time. The three scenarios simulated were: baseline, baseline+reservations, baseline+reservations+policy restrictions. For this particular workload, the baseline queue wait time was 6.03 hours, and expansion factor was 2.05. When reservations were added to the schedule, there was a drastic increase in queue wait time to 74.0 hours. Expansion factor also increased to 7.35.

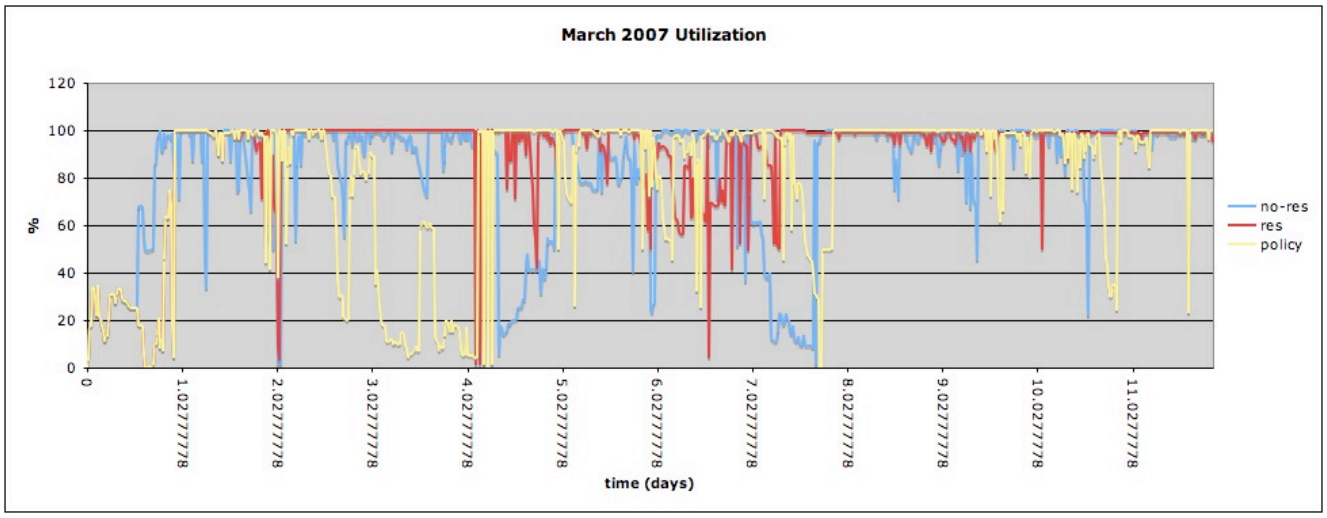
This would be expected, since jobs were scheduled around the obstructing reservations. Some jobs would probably be delayed, resulting in longer queue wait times. When restricting policies were added, to throttle requested reservations, queue wait time dropped to 4.44 hours and expansion factor went to 1.70. This was unexpected, since these metrics are even better than in the baseline, no-reservations scenario. A plausible explanation for this effect is that wide reservations (those spanning much of the machine) create backfill or afterfill windows that allow large jobs to run more quickly than without reservations in place. The reservations would have the effect of preventing small, long jobs from blocking large jobs. In fact, when looking at the simulation data in detail, it became clear that two injected reservations, one for 130 nodes and one for 128 nodes, created a space in the schedule for a 256-node job to run earlier than it would have in the absence of reservations. This case shows that some combinations of workload and reservations may result in increased efficiency.

For the March 2007, 706-job workload, a different pattern occurred. The baseline queue time was 13.7 hours, with expansion factor at 2.53. When reservations were added, queue time went to 107 hours, and expansion factor went to 9.35. With throttling policies, queue wait time recovered to 44.0 hours, and expansion factor went to 5.21. With this workload, reservations had a large negative impact on scheduling metrics. The throttling policies were able to reduce this effect, but scheduling was still less efficient than the baseline case.

One factor to consider is that the simulation does not enforce max jobs per user queued policies, so there can be a hundred one-node, 30 minute jobs waiting to run, giving a very poor expansion factor. On DataStar, we limit the users to six jobs queued at a time, so you wouldn't see the poor expansion factor due to queue stuffing. We would need to examine the individual job expansion factors to know if that's what's going on in the simulation.

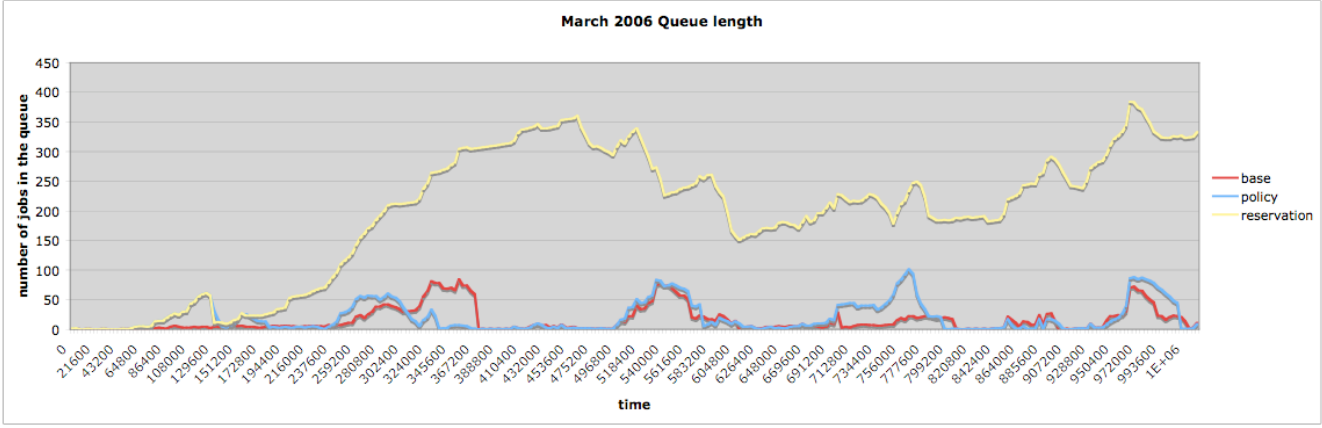


Graph 6. March 2006 Utilization.

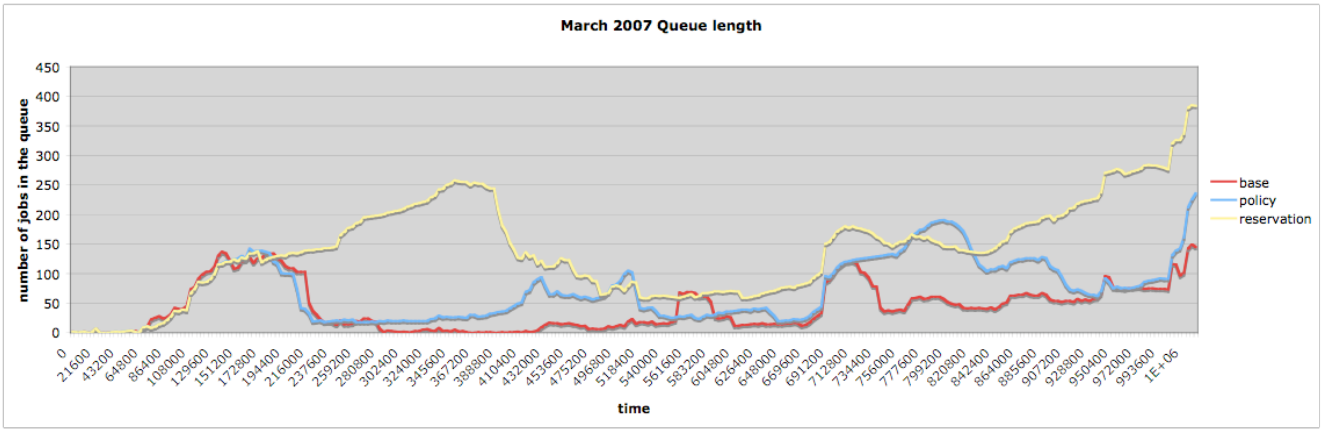


Graph 7. March 2007 Utilization.

Utilization percentage snapshots were taken throughout the simulations. When reservations are counted as utilized parts of the machine, overall utilization remains high. There are some periods in which reservations or reservations+policy results in lower utilization than baseline, but these are not the rule.

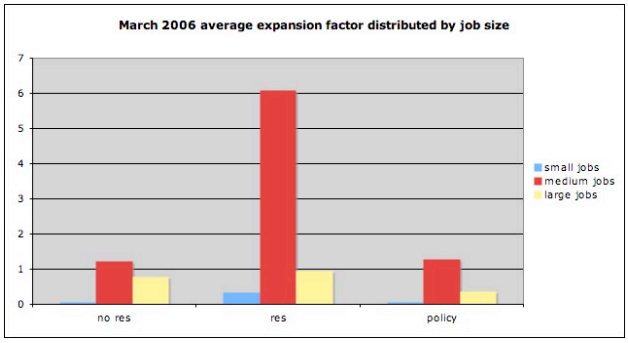


Graph 8. March 2006 Queue Length.

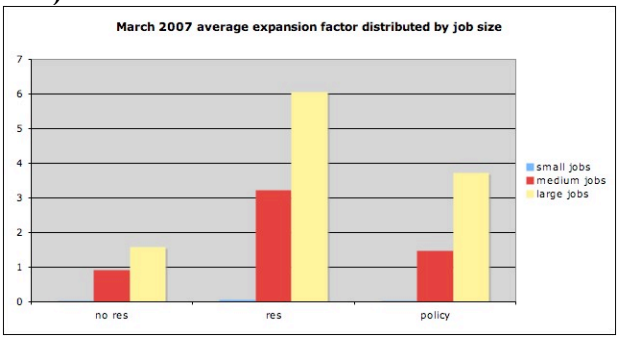


Graph 9. March 2007 Queue Length.

To address the effect of a long job queue on user job submission behavior [13], we examined the length of the job queue as the simulation progressed. Graphs of the queue depth throughout the simulation suggest that in the baseline and baseline+reservations+policy scenarios the queue depth stayed within a realistic regime: 90 waiting jobs for 2006, 150 waiting jobs for 2007. The baseline + reservations queue depth rises much higher than the other two scenarios. Users might be expected to stop submitting jobs before the queue depth reaches that point. The baseline+reservation queue depth may represent a worst-case scenario for expansion factor and queue wait time, in which users continue to blindly submit jobs regardless of queue depth.

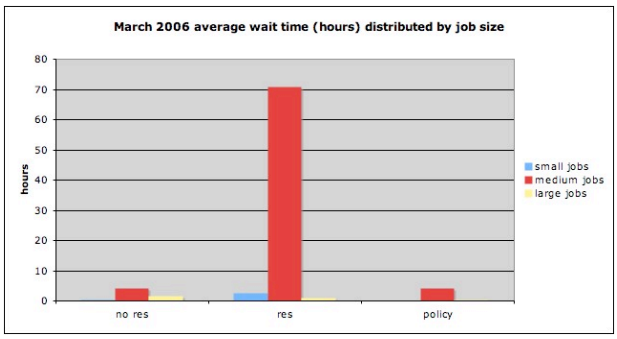


Graph 10. 2006 Expansion factor (by job size) .

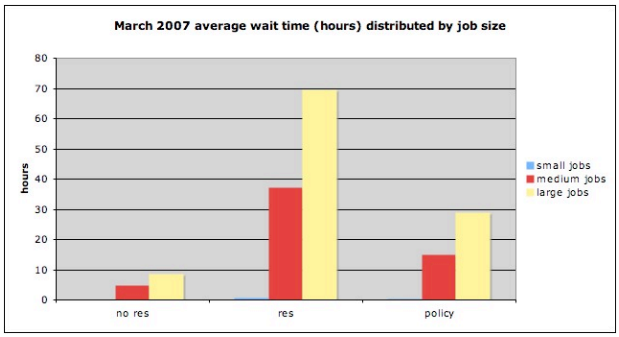


Graph 11. 2007 Expansion factor (by job size).

We wanted to breakdown the components of our expansion factor and wait time changes by job size. The small jobs are one-node jobs, the medium jobs are 2-64 nodes and the large are 65-256. This would allow us to determine whether different size classes of jobs were disproportionately affected by injection of reservations. In 2006, medium-sized jobs suffered more as measured by an expansion factor increase, especially in the baseline+reservations case. In 2007, the expansion factors of large jobs increased over the 2006 simulation. In this simulation, the overall expansion factor with policy did not recover to the baseline case.



Graph 12. Average Wait Time for 2006 Based on Job Size.



Graph 13. Average Wait Time for 2007 based on Job Size.

In 2006, medium-sized jobs suffered more as measured by an increase in average wait time, especially in the baseline+reservations case. In 2007, the average expansion factor and the average wait time for both medium and large jobs suffered in the baseline+reservations scenario. Even after applying our throttling policy, medium and large jobs did not recover expansion factor and queue wait time to the baseline case. In general, medium jobs can be expected to suffer more than small-short or large jobs. In a single-queue, priority-based, reservations scheduling system, medium jobs may be too small to be granted top priority in the queue and may be too large to fit in natural backfill windows. We see here that the effect on large jobs is dependent on the workload, since the March 2006 and March 2007 workloads displayed a different effect of reservations on large jobs.

7. Future Work

In the next few months we will develop hypotheses on the effects of reservations as a function of reservation size, duration, and advance notice. From these hypotheses we will fashion several postulated policies that attempt to provide the advantages of advanced reservations while reducing the deleterious effects on utilization, queue waits, and expansion factors to a minimum. Indeed, it is possible, that, dependent on the details of the job mix, it may be possible to improve some of these in many situations with an appropriate policy.

8. Conclusions

By running two different production job workloads (from DataStar, March 2006 and March 2007) through three different reservation scheduling scenarios (baseline, baseline+reservations, baseline+reservations+policy), we show that injecting reservations can have a large negative impact on average expansion factor and average queue wait time. This effect can be moderated by policies that throttle reservation injection. System utilization was not necessarily worse when reservations were added to the system. The utilization appears to be very dependent on job mix. One concern with simulating the workload was the effect of extremely deep job queues on user psychology. In the case of baseline+reservations, queue depths could run to almost 400 jobs in the queue. Would users continue to submit jobs

under that circumstance? On the other hand, with throttling policies in place, queue depth was comparable to the baseline scenario.

One of the most interesting aspects of this work is that the introduction of advance reservations can, under some circumstances, be advantageous to the existing job mix. The effects of the reservations come from three areas: its "width", or number of nodes, the impact of its "leading edge", and that of its "trailing edge". For simplicity, we can consider the effects of machine-wide reservation, where all nodes are requested (a not unusual scenario, since most reservations tend to be significantly larger than normal jobs). The machine must be drained for the reservation to begin, which naturally tends to produce "holes" in the job mix, reducing utilization. However, when the reservation ends, the job scheduler has had an increased time to try to put an efficient set of jobs together to fully utilize the empty machine at the end of the reservation. Indeed, users will have been continually submitting jobs for the duration of the reservation, increasing the job pool available to the scheduler. This can lead to increased utilization immediately following the reservation.

It is clear that while the deleterious effects of the reservations "leading edge" are independent of the duration of the reservations, the advantageous effects of the "trailing edge" increase with the duration of the reservation. Thus, the effects of allowing reservations can be moved towards the positive by policies that have a minimum duration requirement. We will investigate this in future work. The "width" of the reservation can have similar effects, but these are dependent on the job mix on the particular machine being used. Obviously the history of the job mix is available to the local scheduler: it may be possible to feed back that information into advance reservation policies to again reduce the negative impact.

9. References

[1] D. Feitelson, L. Rudolph, U. Schwiegelshohn, "Parallel Job Scheduling – A Status Report", 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004.

[2] J. Singaga, H. Mohamed, D. Epema, "Dynamic Co-Allocation Service in Multicluster Systems", 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004.

[3] A. Sodan, L. Lan, "LOMARC – Lookahead Matchmaking for Multi-Resource Coscheduling", 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004.

[4] Q. Snell, M. Clement, D. Jackson, C. Gregory, "The Performance Impact of Advance Reservation Meta-scheduling", Job Scheduling Strategies for Parallel Processing: IPDPS 2000 Workshop, JSSPP 2000, Cancun, Mexico, May 2000, Proceedings, Lecture Notes in Computer Science vol. 1911, pp. 137-153, 2000.

[5] San Diego Supercomputer Center at UCSD, http://www.sdsc.edu

[6] IBM LoadLeveler, http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp

[7] Catalina scheduler, http://www.sdsc.edu/catalina

[8] TeraGrid, http://www.teragrid.org

[9] Neutron source, http://en.wikipedia.org/wiki/Spallation_Neutron_Source

[10] P. Andrews, C. Jordan, P. Kovatch, "Massive High-Performance Global File Systems for Grid Computing", Research Paper, SuperComputing 2005, Seattle, WA, November 2005.

[11] K. Yoshimoto, P. Kovatch, P. Andrews, "Co-scheduling with User-Settable Reservations", 11th Workshop on Job Scheduling Strategies for Parallel Processing, 2005.

[12] Metascheduling Requirement Analysis Team final report, http://www.teragridforum.org/mediawiki/images/b/b4/MetaschedRatReport.pdf

[13] E. Frachtenberg, DG. Feitelson, "Pitfalls in Parallel Job Scheduling Evaluation", Job Scheduling Strategies for Parallel Processing, Springer, 2005.

[14] SimPy, http://simpy.sourceforge.net/