Scheduling with Advanced Reservations

Warren Smith[*†] Ian Foster[*] Valerie Taylor[†]

[*]*Mathematics and Computer Science Division*
Argonne National Laboratory, Argonne, IL 60439
{wsmith,foster}@mcs.anl.gov

[†]*Electrical and Computer Engineering Department*
Northwestern University, Evanston, IL 60208
taylor@ece.nwu.edu

Abstract

Some computational grid applications have very large resource requirements and need simultaneous access to resources from more than one parallel computer. Current scheduling systems do not provide mechanisms to gain such simultaneous access without the help of human administrators of the computer systems. In this work, we propose and evaluate several algorithms for supporting advanced reservation of resources in supercomputing scheduling systems. These advanced reservations allow users to request resources from scheduling systems at specific times. We find that the wait times of applications submitted to the queue increases when reservations are supported and the increase depends on how reservations are supported. Further, we find that the best performance is achieved when we assume that applications can be terminated and restarted, backfilling is performed, and relatively accurate run-time predictions are used.

1. Introduction

A recent trend in high-performance computing are computational grids [5]. Computational grid applications use high-performance, distributed resources such as computers, networks, databases, and instruments. Such applications are enabled by grid toolkits such as Globus [4] or Legion [6] that provide a software infrastructure for security, information, resource management, communication, access to remote data, and other services that are typically layered over existing local services. Several computational grid testbeds have been deployed [1, 2, 7, 13, 14] and we have found that many applications have very large resource requirements and require resources from multiple parallel computers to execute.

The difficulty with these applications is that current supercomputer scheduling systems do not provide mechanisms that allow several scheduling systems to provide simultaneous access to resources. At the present time, a user has to either communicate with the administrators of the computers and arrange for resources to be reserved, or submit applications to queues on each computer system with no guarantee that the subapplications will execute simultaneously. In this paper, we investigate one solution to this *co-allocation* problem: advanced reservation of resources. Reservations allow a user to request resources from multiple scheduling systems at a specific time and thus gain simultaneous access to enough resources for their application. Advanced reservations are currently being added to the Portable Batch System (PBS) [15] and the Maui scheduler [12], but a thorough study of the implications of supporting for advanced reservations in scheduling systems has not been made.

We investigate several different ways to add support for reservations into scheduling systems and evaluate their performance. We evaluate scheduling performance using the following metrics:

- *Utilization.* The average percent of the machine that is used by applications.
- *Mean wait time.* The average amount of time that applications wait before receiving resources.
- *Mean offset from requested reservation time.* The average difference between when the users initially want to reserve resources for each application and when they actually obtain reservations.

The utilization and mean wait time metrics allow us to examine the effect that support for reservations has on traditional scheduling performance. The mean offset from requested reservation time is a new metric and measures how well the scheduler performs at satisfying reservation requests.

In this paper, we use these metrics to evaluate a variety of techniques for combining scheduling from queues with reservation. There are several assumptions and choices to be made when doing this. The first is whether applications are restartable. Most scheduling systems currently assume that applications are not restartable (a notable exception is the Condor system [9]). We evaluate scheduling techniques when applications both can and cannot be restarted. We assume that when an application is terminated, intermediate results are not saved and applications must restart execution from the beginning. We also assume that a running application that was reserved cannot be terminated to start another application. Further, we assume that once the scheduler agrees to a reservation time, the application will start at that time. Further details of our model along with other background information are presented in Section 2

If we assume that applications are not restartable and that once a reservation is made, the scheduler must fulfill it, then we must use maximum run times when predicting application execution times to ensure that nodes are available. The resulting scheduling algorithms essentially perform backfilling. Maximum run times are typically either given by the user for each application or are associated with a queue. If an application executes longer than it's maximum run time it may be terminated. Details of scheduling algorithms making this assumption and an evaluation of their performance are presented in Section 3.

If applications are restartable, there are more options for the scheduling algorithm and this allows us to improve the scheduling performance. First, the scheduler can use run-time predictions other than maximum run times. Second, there are many different ways to select which running applications from the queue to terminate to start a reserved application. Details of these options and their performance are presented in Section 4

2. Background

This section describes the scheduling algorithms we modify to support reservations, the workloads we use to evaluate our algorithms, and the model we use for reservations.

2.1. Scheduling Algorithms

We modify scheduling algorithms that use two different queue orders and that may or may not perform backfilling. The two basic queue orders are first-come first-served (FCFS) and least work first (LWF). For FCFS, applications are ordered by the time in which they arrive. For LWF, applications are ordered by the predicted amount of work they will perform (number of nodes multiplied by estimated wallclock execution time). We also apply conservative backfilling [8, 3] to both of these queue orderings. The backfill algorithm allows an application to run before it would in it's queue order if it will not delay the execution of applications ahead of it in the queue.

2.2. Workloads

We begin with four workloads recorded from three supercomputers to evaluate our scheduling algorithms. The workload traces that we consider are described in Table 1; they originate from Argonne National Laboratory (ANL), the Cornell Theory Center (CTC), and the San Diego Supercomputer Center (SDSC).

To evaluate our different scheduling algorithms that support reservations, we derive two new workloads from each of the four workloads described in Table 1. We randomly change either 10 percent or 20 percent of the applications in an original workload to be reservations. We choose these percentages because we believe that the majority of applications will not need reservations to execute and policy decisions will be made so that there will be no start time advantage to making a reservation over submitting to a queue. For each reservation, we randomly set the requested reservation time to be within zero to two hours in the future. We also experimented with setting the requested reservation time to be one to three or two to four hours in the future [10]. We found that the performance was almost identical for the three ranges so we only present results where reservations are requested zero to two hours in advance.

2.3. Reservation Model

Next, we describe the model we use for reservations. First, in our model, a reservation request consists of the number of nodes desired, the maximum amount of time the nodes will be used, the desired start time, and the application to run on those resources. Second, we assume that the following procedure occurs when a user wishes to submit a reservation request:

Table 1. Characteristics of the workloads used in our studies.

Workload Name	System	Number of Nodes	Location	When	Number of Requests	Mean Run Time (minutes)
ANL [1]	IBM SP2	120	ANL	3 months of 1996	7994	97.40
CTC	IBM SP2	512	CTC	11 months of 1996	79302	182.18
SDSC95	Intel Paragon	400	SDSC	12 months of 1995	22885	107.76
SDSC96	Intel Paragon	400	SDSC	12 months of 1996	22337	166.48

1. The user asks if they can run an application at time T_r on N nodes for at most M amount of time.

2. The scheduler makes the reservation at time T_r if it can. In this case, the reservation time, T, equals the requested reservation time, T_r.

3. If the scheduler cannot make the reservation at time T_r, it replies with a list of times it could make the reservation and the user picks the available time T which is closest in time to T_r.

The last part of the model is what occurs when an application is terminated. First, only applications that came from a queue can be terminated. Second, when an application is terminated, it is placed back in the queue from which it came in its correct position.

3. Nonrestartable Applications

In this section, we assume that applications cannot be terminated and restarted at a later time and that once a reservation is agreed to by the scheduler, it must be fulfilled. A scheduler with these assumptions must not start an application from a queue unless it is sure that starting that application will not cause a reservation to be unfulfilled. Further, the scheduler must make sure that reserved applications do not execute longer than expected and prevent other reserved applications from starting.

There are two mechanisms to be described to support these constraints. The first is how the scheduler decides when an application from a queue can be started. The technique used for this is very similar to the backfill algorithm: The scheduler creates a timeline of when it believes the nodes of the system will be used in the future. First, the scheduler adds the currently running applications and the reserved applications to the timeline using their maximum run times. Then, the scheduler attempts to start applications from the queue using the timeline and the number of nodes and maximum run time requested by the application to make sure that there are no conflicts for node use.

If backfilling is not being performed, the timeline is still used when starting an application from the head of the queue to make sure that the application does not use any nodes that will be needed by reservations. If backfilling is used, the timeline is used to try to start applications from the queue and to "reserve" nodes for the applications at the earliest time in the future that they can run if they cannot start at the current time. These "reservations" are not true reservations, just placeholders so that applications later in the queue will not start and delay the application.

The second mechanism is how a scheduler makes a reservation. To make a reservation, the scheduler first performs a scheduling simulation of applications currently in the system and produces a timeline of when nodes will be used in the future. This timeline is then used to determine when a reservation for an application can be made. The scheduler uses maximum run times when creating the timeline. This guarantees that reserved applications do not conflict with running applications or other reserved applications.

One parameter that is used when reserving resources is the relative priorities of queued and reserved applications. For example, if queued applications have higher priority, then an incoming reservation cannot delay any of the applications in the queues from starting. If reserved applications have higher priority, then an incoming reservation can delay any of the applications in the queue. The parameter we use is the percentage of queued applications can be delayed by a reservation request and this percentage of applications in the queue is simulated when producing the timeline that defines when reservations can be made.

In the next subsections, we first examine the effect reservations have on utilization and the mean wait time

[1] Because of an error when the trace was recorded, the ANL trace does not include one-third of the requests actually made to the system. To compensate, we reduced the number of nodes on the machine from 120 to 80 when performing simulations.

of applications from the queue. Second, we examine the changes in the difference between the requested reservation time and the time the reservation is actually made when different scheduling strategies are used. Third, we look at the changes in performance when we vary the number of applications from the queue that an application can delay.

3.1. Effect of Reservations on Scheduling

This section evaluates the impact on the mean wait times of queued applications when reservations are added to our workloads. We assume the best case for queued applications: When reservations arrive, they cannot be scheduled so that they delay any currently queued applications. First, we examine the wait times of queued jobs when backfilling is not allowed.

We find that adding reservations increases the wait times of queued applications in almost all cases. For all of the workloads, queue wait times increase an average of 13 percent when 10 percent of the applications are reservations and 62 percent when 20 percent of the applications are reservations. Our data also shows that if we perform backfilling, the mean wait times increase by only 9 percent when 10 percent of the applications are reservations and 37 percent when 20 percent of the applications are reservations. This is a little over half of the increase in mean wait time when backfilling is not used. Further, there is a slightly larger increase in queue wait times for the LWF queue ordering than for the FCFS queue ordering.

We also examined the utilization of the machines being simulated for our various experiments. We find that the utilization does not change for the CTC and SDSC workloads for any queue ordering, backfilling, or any number of reservations. This occurs because of the workloads themselves. The applications in the workloads arrive steadily over time with more submissions occuring during the day and less at night. Even the most inefficient scheduling algorithm studied here does not fall far enough behind the arriving jobs so that it takes a significant amount of time longer to finish executing all of the jobs in the workload.

At first, the results for the ANL workload appear to be different. If no reservations are made, then the utilization is between 70 and 71 percent (the highest of any of our workloads) for both queue orderings and if backfilling is or is not used. Once again, this is demonstrating that even the most inefficient scheduling algorithm does not fall very far behind arriving jobs, even for our most demanding workload. But, when reservations are supported, the utilization drops to between 54 and 59 percent. The utilizations are higher when there are 10 percent reservations instead of 20 percent. This seems to indicate that support for reservations will have a large effect on the utilization of highly-loaded systems, but closer examinination of the data contradicts this theory. The data shows that the lower utilization is due a few reserved applications at the end of the simulations If these last reserved applications are not considered, then the utilization only decreases slightly when reservations are supported. We claim that the affect on utilization of these last reservations can be ignored because in a real computer system, there is not end to the workload being scheduled.

3.2. Offset from Requested Reservations

In this section, we examine the difference between the requested reservation times of the applications in our workload and the times they receive their reservations. We again assume that reservations cannot be made at a time that would delay the startup of any applications in the queue at the time the reservation is made.

The performance is what is expected in general: the offset is larger when there are more reservations. For 10 percent reservations, the mean difference from requested reservation time is 211 minutes. For 20 percent reservatons, the mean difference is 278 minutes. This is an increase of 32 percent over the mean difference from requested reservation time when 10 percent of the applications are reservations.

Our data also shows that the difference between requested reservation times and actual reservation times is 49 percent larger when FCFS queue ordering is used. The reason for this may be that LWF queue ordering will execute the applications currently in the queue faster than FCFS. Therefore, reservations that cannot delay any queued jobs can start earlier.

We also observe that if backfilling is used, the mean difference from requested reservation times increases by 32 percent over when backfilling is not used. This is at odds with the previous observation that LWF queue ordering results in smaller offsets from requested reservation times. Backfilling also executes the applications in the queue faster than when there is no backfilling. Therefore, you would expect a smaller offsets from requested reservation times. An explanation for this behavior could be that backfilling is packing applications from the queue tightly onto the nodes and is not leaving many gaps free to satisfy reservations before the majority of the applications in the queue have started.

3.3. Effect of Application Priority

Next, we examine the effects on mean wait time and the mean difference between reservation time and requested reservation time when queued applications are not given priority over all reserved applications. We accomplish this by giving zero, fifty, or one-hundred percent of queued applications priority over a reserved application when a reservation request is being made (rephrased, not delaying one-hundred, fifty, or zero percent of queued applications when a reservation is made).

The data shows that there is a significant impact on both wait time and offset from requested reservation time when the number of queued applications that can be delayed by reservations is varied. As expected, if more queued applications can be delayed when a reservation request arrives, then the wait times are generally longer and the offsets are smaller. On average, for the ANL workload, decreasing the percent of queued applications with priority from 100 to 50 percent increases mean wait time by 7 percent and decreases mean offset from requested reservation times by 39 percent. Decreasing the percent of queued application with priority from 100 to 0 percent increases mean wait time by 22 percent and decreases mean offset by 89 percent. These results for the change in the offset from requested reservation time are representative of the results from the other three workloads: as fewer queued applications have priority, the reservations are closer to their requested reservations.

The large decrease in mean offset from requested reservation time when the number of queued jobs with priority decreases compared to the smaller increases in mean wait times seems to indicate that reservations should be given priority over queued jobs. The difficulty with this approach is that users would notice this and would therefore start making reservations for applications that could have been sent to the queue. This would increase the percent of applications that are reservations and, our data shows, increase the average wait time of all applications.

4. Restartable Applications

This section describes and evaluates our techniques for performing reservations assuming that running applications can be terminated and restarted at a later time. If we make this assumption, we can use run-time predictions other than maximum run times and this allows us to improve scheduling performance.

4.1. Run-Time Predictions

We use a technique that we have previously developed [11, 10] to predict the execution times of applications. This technique uses a historical database of applications that have executed in the past to find similar applications and to derive run-time predictions.

4.2. Selecting Applications for Termination

There are many possible ways to select which running applications that came from a queue should be terminated to allow a reservation to be satisfied. We choose a rather simple technique where the scheduler orders running applications from queues in a list based on some cost. The applications are then terminated in increasing order of cost until enough nodes are available for the reservation to be satisfied.

We use the equation $aNT_p + bNT_f$ to determine the cost of terminating each application. In the equation, a, and b are constants, N is the number of nodes being used by the application, T_p is the amount of time the application has executed, and T_f is the amount of time the scheduler expects the application will continue to execute. The motivation behind this equation is that increasing the constant a will increase the cost of terminating an application that has performed a large amount of work that would be lost, and decreasing b below zero will decrease the cost of terminating the application if it still has a large amount of work to do.

We vary the constants a and b to determine the optimal values for a and b. We choose a and b such that $a - b = 1.0$ and vary a between 0.0 and 1.0 in increments of 0.1. These values allow us to perform experiments varying the percentage of the termination cost associated with the amount of work performed from zero to one hundred percent with the amount of work yet to do contributing the remaining percent. Our data shows that the best values to use for the constants vary by the scheduling algorithm and if the mean wait time or mean difference from requested reservation time is being optimized. However, there are several trends that can be seen in the data. First, $|a|$ is larger than $|b|$. This indicates that the amount of work done thus far is the most important factor to consider when selecting which applications to terminate. Second, in over half of the cases both mean wait time and the mean difference in reservation is optimized with the same values of a and b. Third, we observe that the mean wait times do not change smoothly as, say, a is increased from 0.0 to 1.0.

4.3. Comparison to Nonrestartable Techniques

We will now compare scheduling performance when applications can be terminiated to when they cannot. We performed simulations using only the ANL workload due to time constraints. Our data shows that if applications can be terminated and restarted, the mean wait time decreases by 7 percent and the mean difference from requested reservation time decreases by 55 percent. There is no significant effect on utilization. This shows that there is a performance benefit if we assume that applications are restartable, particularly in the mean difference from requested reservation time.

5. Conclusions

In this paper we examine the performance of several different techniques for combining scheduling using queues with reservations. First, we examine techniques when applications cannot be restarted. We find that this forces us to use maximum run times for run-time predictions and techniques similar to backfilling. If we assume that reservations cannot delay the start of any of the applications in the queue when a reservation is made, then supporting reservations with backfilling increases the wait times of applications in the queue by 9 percent when 10 percent of the applications are reservations and by 37 percent when 20 percent of the applications are reservations. We also find that the mean difference between requested reservation times and reservation times is 211 minutes when 10 percent of the applications are reservations and 278 minutes when 20 percent of the applications are reservations. We show that for the ANL workload (which is representative) if we decrease the percent of queued applications that cannot be delayed by a reservation from 100 to 50 then the mean wait time increases by an average of 7 percent and the mean difference from the requested reservation time decreases by 39 percent. If we decrease the percent of queued applications with priority from 100 to 0 percent then the mean wait time increases by 22 percent and the mean difference decreases by 89 percent.

Second, we evaluate scheduling techniques that assume that applications can be terminated and restarted at a later time. We use an equation to determine the cost of terminating each running application and use these costs when picking applications to terminate. We find that the cost should largely be determined by the amount of time the application has executed and the number of nodes it has used, but a prediction on the amount of time the execution has left to run should also be considered. Finally, if we assume that applications can be restarted and therefore our run-time predictions can be used, the mean wait time is decreased by 7 percent on average and the mean difference between the requested reservation times and the actual reservation times decreases by 55 percent.

References

[1] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metasystems. *Lecture Notes on Computer Science*, 1998.

[2] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, 1996 (to appear).

[3] D. Feitelson and A. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, 1998.

[4] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann, 1999.

[6] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. R. Jr. Legion: The Next Logical Step Toward a Nationwide Virtual Computer. Technical Report CS-94-21, University of Virginia, June 1994.

[7] W. Johnston, D. Gannon, and B. Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.

[8] D. A. Lifka. The ANL/IBM SP Scheduling System. *Lecture Notes on Computer Science*, 949:295–303, 1995.

[9] M. Litzkow and M. Livny. Experience With The Condor Distributed Batch System. In *IEEE Workshop on Experimental Distributed Systems*, 1990.

[10] W. Smith. *Resource Management in Metacomputing Environments*. PhD thesis, Northwestern University, December 1999.

[11] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times Using Historical Information. *Lecture Notes on Computer Science*, 1459:122–142, 1998.

[12] The Maui Scheduling System. http://www.mhpcc.edu/maui.

[13] The National Computational Science Alliance. http://www.ncsa.uiuc.edu/alliance.

[14] The National Partnership for Advanced Computing Infrastructure. http://www.npaci.edu.

[15] The Portable Batch System. http://pbs.mrj.com.