

The Performance Impact of Advance Reservation Meta-scheduling

Quinn Snell, Mark Clement, David Jackson, and Chad Gregory

Brigham Young University, Provo, Utah 84602
{snell, clement, gregory}@cs.byu.edu jacksond@mhpcc.edu

Abstract. As supercomputing resources become more available, users will require resources managed by several local schedulers. To gain access to a collection of resources, current systems require metajobs to run during locked down periods when the resources are only available for metajob use. It is more convenient and efficient if the user is able to make a reservation at the soonest time when all resources are available. System administrators are reluctant to allow reservations external to locked down periods because of the impact reservations may have on utilization and the Quality of Service that the center is able to provide to its normal users. This research quantifies the impact of advance reservations on and outlines the algorithms that must be used to schedule metajobs. The Maui scheduler is used to examine metascheduling using trace files from existing supercomputing centers. These results indicate that advance reservations can improve the response time for metajobs, while not significantly impacting overall system performance.

1 Introduction

Recently, there have been a number of research groups focusing their efforts on utilizing the combined resources of multiple supercomputing facilities [1–4]. The motives for this avenue of research are obvious. First, and perhaps foremost, is the belief that the proper combination of resources and their aggregate processing power will yield a system that is both scalable and potentially more efficient. A system such as the one proposed in this paper would provide enhanced scheduling services for at least three kinds of jobs: those that require more resources than are available at any one site, jobs that require a combination of resources that are not available at any one site, and finally, jobs from users that desire a better overall response time than could be obtained by limiting their jobs to using the resources found at any single site.

Regardless of the job type, using the combined resources of more than one site requires cooperation that is not inherent in local resource scheduling systems. A system that coordinates and works with the local schedulers is required. Such a system is often called a metascheduler. The metascheduler makes this aggregation of resources available to what are termed throughout this paper as 'metajobs'. Each metascheduler maintains a job queue to which these metajobs are submitted by users where they are stored until they complete. A metajob can

be simply a normal batch job submitted to the meta scheduler or it may be modified to utilize special functionality that can only be found in a metascheduling environment. There are three key differences between a metajob and a standard batch job: 1) The machine or local scheduler under which the metajob will run is not known at job submit time. 2) The metajob may contain a utility function which instructs the metascheduler as to what aspects of the resources are most important to it. These aspects may include cost, machine speed, time until availability, etc. 3) The resources utilized by the metajob may span more than one machine or local scheduler.

Some current metascheduling systems work by dedicating a set of local resources to be used and scheduled by the metascheduler. With these metaschedulers, local management and administration staff determine both a maximum amount of resources allowed for metascheduled work and a set of timeframes during which these resources can be used. During each of these timeframes, the metascheduler assumes full control of all allowed resources, preventing their use by locally submitted jobs. These resources remain unavailable regardless of the metascheduled workload. Supercomputer system managers have reported average utilizations ranging between 5% and 25% on those nodes that are dedicated for metacomputing systems. Due to the fragmentation of resources, low utilizations occur even when there is a backlog of local jobs. A metascheduling system based on this model clearly wastes valuable resources, significantly lowering the overall system utilization and increasing the average job turnaround time for local jobs.

This dedicated resource metascheduler schedules resources as if it were a local scheduler, according to its own private set of policies and priorities. It does not leverage the knowledge or capabilities of the local scheduler. The local scheduler has, in reality, completely relinquished control of the resources that have been dedicated to the metascheduler. This brings up more issues. First, lower utilization can be expected since the local compute resources are now fragmented. Additionally, the resource fragments are exclusive of each other meaning that those resources dedicated for metascheduled jobs cannot be used by locally submitted jobs even if they are idle; the same holds true for metascheduled jobs and resources dedicated for local workload. Also, under existing 'dedicated resource' metaschedulers, each resource fragment is scheduled according to an independent, private set of policies. There is no cooperation between the local schedulers and metaschedulers, and there is no knowledge of each others policies, priorities, or workload. In consequence of these conditions, there is no opportunity for cross-fragment scheduling optimizations such as backfill or intelligent node allocation.

Ursala, a metascheduler developed at Brigham Young University and used for the research described in this paper, cooperates with the local schedulers and introduces metascheduled jobs into the locally-produced workload. A system capable of doing this can achieve better overall utilization of resources because full information *and* control are maintained at the local scheduler level. Local scheduling optimizations such as backfill can then occur. Instead of ded-

indicating the maximum block of resources allowed by the administrator, Ursala intelligently reserves only those resources that are required for existing, queued metajobs.

Ursala operates by breaking up a metajob into a number of localized sub-jobs, each of which is submitted to a different local resource manager. With parallel jobs, it is imperative that all resources required by the entire job be made available at job start time. The metascheduler must be able to determine a time when all of the needed resources are available. Each local scheduler must also be able to guarantee the availability of the needed resources for the corresponding sub-job at that determined time.

The guarantee of providing specific resources at a specific time is typically termed an advance reservation. Foster *et. al* [5] report that currently there is “no widely deployed resource management system for networks, computers, or other resources” that supports advance reservations. This is no longer the case. The Maui scheduler[6, 7] has extensive support for advance reservations and is the local scheduler used in this research.

Creating coinciding advance reservations on each of the needed systems is not as simple as using a block of dedicated resources. However, the advantages of using such a system are tremendous. First, there is no fragmentation and there are no resources sitting idle while the metascheduled workload is low. The metascheduler only reserves the resources it needs exactly when it needs them. Thus, the local scheduler is allowed to utilize these resources at all other times and the overall utilization of the system is consequently higher. Each local scheduler is also able to enforce its local policies for all jobs. There are still questions which must be answered about a metascheduler that uses advance reservations to launch jobs across multiple systems: “What effect does metascheduling using advance reservations have on overall system utilization?”, “What problems arise when trying to create coinciding reservations?”, and “What impact does this type of metascheduling have on both local and metajob turn around time?”

This paper will deal with the issues listed above. Sections 2 and 3 discuss advance reservations and metascheduling respectively. The issues that surround metascheduling based on advance reservations are then discussed in Section 4. Finally, we present our metascheduling system and the experimental results we have obtained to answer the questions posed above.

2 Advance Reservations

In the most general sense, an advance reservation is a scheduling object which reserves a group of resources for a particular timeframe for access only by a specified entity or group of entities. These configurable aspects of an advance reservation can be set to support a metajob. The settings required are listed below.

1. A reservation must reserve exactly the type and amount of resources requested by the job.

2. A reservation must reserve these resources for use at the requested start time.
3. A reservation must reserve these resources for the wallclock limit of the metajob.
4. A reservation must reserve these resources for use only by the specified metajob.
5. The local scheduler must guarantee that the metajob will only run in the job reservation even if resources exist elsewhere which would allow the job to run earlier.

Reservations which meet the above configuration are termed a *job* reservation. While local schedulers may support advance reservations with additional attributes and features, the attributes listed above are the minimal set required to properly reserve resources for a metajob.

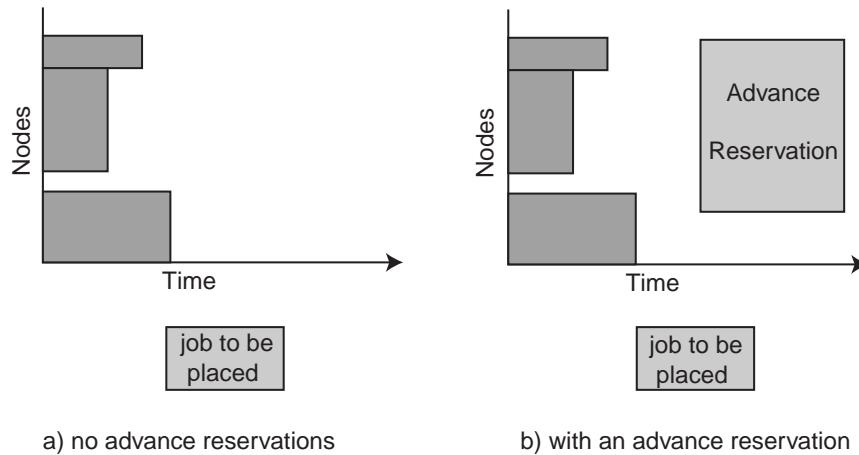


Fig. 1. Comparison of scheduling with and without advance reservations

Scheduling in the presence of advance reservations is significantly more complex than normal scheduling. Without advance reservations, a scheduler simply sets a policy of not scheduling any job that will delay the start time of the highest priority job in the queue. As long as that policy is maintained, the scheduler is free to take jobs from other parts of the queue and backfill the nodes. The scheduler need only consider if adequate resources exist at the present time to start a given job and need not determine if a job will fit on a given set of nodes in the time dimension.

Scheduling with advance reservations and guaranteed start time requires that the scheduler be capable of not only fitting jobs into the node space dimension, but also fitting them into the time dimension. This potentially involves analyzing resource dedication to other jobs both before and after the time of interest.

Figure 1 compares scheduling with and without advance reservations. Note that without advance reservations, the future in the time dimension is open. With only a single advance reservation in place, the scheduler must now consider if a job will have enough resources available and if the resources will be available long enough for the job to complete before the resources are required by the future reservation. This is a much more complex decision.

Despite the added complexity, there are numerous advantages associated with advance reservations. Once reservations are incorporated, a scheduler can perform reservation-based deadline scheduling. The scheduler can guarantee the start time of a given job so that job start time can be coordinated with data availability. Schedulers can backfill around reservations to minimize the impact of dedicating resources to jobs, users, or projects. These functions yield a higher overall level of service to all users. This paper will show that while advance reservations can be utilized by the local scheduler to provide a number of local services, their greatest value may lie in their use in the metascheduling realm where they can be built upon to provide a new and powerful class of metascheduling services.

3 Meta-scheduling

Meta-scheduling can be loosely defined as the act of locating and allocating resources for a job on a metacomputer. Smarr and Catlett [8] define a metacomputer as the collection of resources that are transparently available to the user via the network. Thus a metascheduling system should make a collection of resources transparently available to the user as if it were a single large system. The key in this definition is that the user need not be aware of where the resources are, who owns the resources, or who administers the resources in order to use them. The scheduling issues that surround the creation of such a system is the focus of this section.

From the point of view of the metascheduler, metajobs fall into two basic categories: those that run on a single machine, and those that span multiple machines. The first category of jobs require less effort on the part of the metascheduler. The metascheduler simply locates which local scheduler can provide the greatest utility to the metajob and submits the metajob into that local scheduler's workload queue. The local scheduler takes care of the rest. Meta jobs that span multiple local schedulers introduce new issues that are not found in local scheduling systems.

The first step in developing a metascheduling system is to define the types of scheduling that will be performed. The following three classifications can be applied to metajobs.

1. Specified Co-Allocation: The user specifies all resource requirements including exactly which processors and resources are required. For example a user may specify that telescope X, 512 IBM SP2 processing nodes from site Y, and an SGI graphics pipe from site Z are needed. The metascheduler would

then find a time when all of the resources were available for the user's priority. In this case, all requests are for particular resources and locations are defined by the user. Reservations are made for each required resource at that time. This is the easiest kind of scheduling and little information need be passed between the scheduler and the metascheduler. Job initialization and resource management issues are all handled locally.

2. **General Co-Allocation:** This type of scheduling differs from specified Co-Allocation in that the user does not specify where to run each part of the job. The user merely specifies the needed resource types and the metascheduler decides when and where the best resources are located, reserves those resources at that time and runs the requested job. For example, the user's specification of a SGI graphics pipe could yield any SGI graphics pipe known to the metascheduler.
3. **Optimal Scheduling:** The extreme case in scheduling is to determine the best location for every resource in the job. Using knowledge about machine and network performance, the metascheduler determines the placement that optimizes cost, performance, response time, throughput, and other factors. This type of metascheduling requires up to date performance knowledge from each part of the metacomputer as well as a characterization of the application's network and CPU requirements. For example, if a user simply requested 100 processors, the scheduler would determine the best 100 processors to use, even if that means fragmenting the job across supercomputer centers.

These three types of scheduling are ordered according to the level of intelligence that is required of the metascheduler. Specified Co-Allocation requires only that the metascheduler optimize in the time dimension, whereas the other categories require that the metascheduler make intelligent decisions in both time and space. Category 2 and category 3 scheduling can be performed by allowing the user to specify a utility function. The metascheduler then locates resources for the job such that the utility function is maximized. This type of scheduling will not be considered here, but is part of ongoing research.

Because all three categories of metascheduling are dependent upon advance reservations, it is critical to determine the impact of these reservations on local workload and their effectiveness in coordinating metajob components. The remainder of this paper will discuss Specified Co-Allocation using advance reservations and their effect on local and metascheduler performance.

3.1 Meta-scheduling using Advance Reservations

As stated above, the goal of metascheduling is to reserve groups of resources from different resource management systems. To accomplish this goal, a metascheduler must create multiple, coinciding advance reservations. Creating these advance reservations can be reduced to the resource allocation problem in distributed algorithms research [9]. It is well known that such problems introduce possibilities of deadlock and livelock. Before discussing these issues, we consider the steps that a metascheduler must take to create coinciding reservations.

1. Determine available resources at each site.
2. Select the resources and time frame.
3. Create the advance reservations at each site.
4. Stage the appropriate job components to each local scheduler.

Determination of the available resources can be performed in many different ways. In the simplest sense, a query is performed on the local scheduler asking if resource set X is available at time Y. A much more flexible metascheduling system may be created if the metascheduler can ask 'what resources are available for job X in the time range A to B?' and the local scheduler responds with all ranges of acceptable times and resource sets that can be made available while still meeting local scheduler policies. For example, '8 nodes available from now until 8 hours out and 16 nodes available from 8 hours out until 24 hours out'. For metajobs which must span local schedulers, the metascheduler must perform an intersection of the returned start time ranges to determine the possible start times. The metascheduler could then automatically select the optimal metajob start time using the job's utility function or present the possibilities to the submitting user for a final decision.

For example, perhaps a scientist would like to allocate resources at three sites during working hours for interactive use. Referring to Figure 2, if the local schedulers can only answer yes or no to a proposed start time, the metascheduler must then propose a new start time if no appropriate times are available. This is shown in the first column of the figure. Flexibility is added when resource availability time ranges are returned as shown in the second column. Now the metascheduler can find the intersecting time ranges and present that to the scientist. However, there may be no appropriate intersecting time, thus forcing the scientist to propose a new start time. If a list of start time ranges is returned, the scientist may see that although there are no appropriate times today, there is a time the next day. This eliminates the need for the scientist to begin the process over again with a new proposed start time.









	yes/no	Range	Range List
System 1	yes		
System 2	no		
System 3	yes		
Result	no	 Time →	 Time →

Fig. 2. Comparison of possible query results returned from local schedulers.

Deadlock and livelock issues are introduced because the metascheduler must gather resource information and make reservations on several local schedulers. The metascheduler receives available time ranges, then determines the appropriate start time and resources for the job before making the reservations. Because there is a brief delay between collecting the resource availability information and making the reservations, it is possible that the availability of the resources under the control of each local scheduler has changed. This could be due to local scheduling or due to reservations created by a competing metascheduler attempting to create advance reservations in the same time frame.

The potential for deadlock is brought about because all of the four deadlock conditions exist [10]. Eliminating one of the four conditions eliminates deadlock. Perhaps the easiest condition to eliminate is Hold and Wait. In a metascheduling environment, this means that as advance reservations are created, if any reservation cannot be made due to recent resource availability changes, all existing reservations for that job must be cancelled. This policy is logical in the case of metascheduling since the local reservations are for a specific time and the job must start on all resources at the same time. Thus deadlock situations are eliminated by forbidding Hold and Wait.

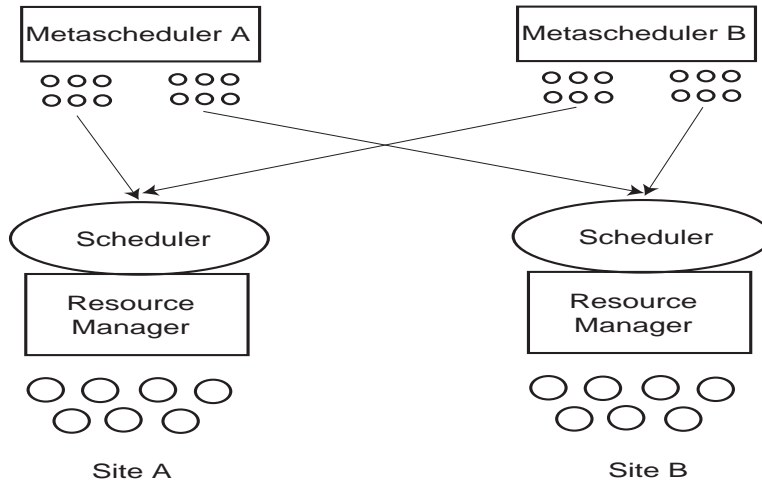


Fig. 3. Livelock scenario. Metaschedulers A and B are competing for resources at sites A and B.

Livelock on the other hand is much more complex. Two separate metascheduling systems can mutually interfere and cause changes in the state of the local schedulers such that each metascheduler is never able to create coinciding reservations on the local machines. For example, consider the scenario depicted in Figure 3. Both metaschedulers need resources at sites A and B. If either site grants the needed resources to one of the metaschedulers, the needed resources

will not be available for the other metascheduler. The resource query and reservation allocation are separate steps. If the metaschedulers proceed through the steps by approaching the local schedulers in opposing order, they will mutually interfere. Both metaschedulers will see that the needed resources are available, and will request reservations from their corresponding sites. However, when the metaschedulers request from opposing sites, they will both fail and release their other reservations. The process will then begin again. If the metaschedulers continue to request, they will most likely get out of sync eventually and one will succeed, thus breaking the livelock. The metaschedulers may go through many of these iterations though.

There are many research areas that have similar livelock characteristics. In particular, the Ethernet [11] CSMA/CD networking protocol has a random exponential backoff policy for dealing with collisions on a shared network. When utilization on the network is high, much of the network bandwidth is spent in collision detection and backoff, resulting in degraded performance. One major difference, however, is that the Ethernet possess only a single shared resource while the metascheduling problem has multiple shared resources, increasing the problem's complexity.

The main cause of the livelock problem is that the local system can change state between the metascheduler's resource query and reservation phases. The state changes can come from two sources, new resource manager information (i.e. a node going down) and new reservations created by competing metaschedulers. While nothing can be done to control actual resource state changes, steps can be taken to eliminate the possibility of resource state due to other metaschedulers. One solution is to have the resource query lock the reservation state of the local scheduler for a period of time preventing other metaschedulers from creating reservations during this time. The locked time would be just long enough to allow the locking metascheduler to process the resource information and make all needed reservations. This locking of reservation state is one form of a *courtesy* reservation. A timeout can be imposed to eliminate the case of a courtesy reservation locking out the entire machine for long periods of time. If the reservation is not made before the timeout, there is no guarantee that the resources will still be available. However, these courtesy reservations introduce several problems into local scheduling algorithms. It is hard to determine the potential effects of courtesy reservations on scheduler performance. Further research is being conducted to examine the tradeoffs between courtesy reservations and stateless reservations combined with a backoff policy.

3.2 The Maui Scheduler

The Maui scheduler is currently in use at many supercomputing facilities and is ideal for this research. The Maui scheduler supports advance reservations in the form previously described allowing the scheduler to guarantee a job's start time. A metascheduling interface allows very flexible resource availability queries to be performed. The metascheduler may specify the job in great detail or it may simply specify the number of resources and the amount of time required. Replies

to these queries are returned as a list of start time ranges and associated resource sets. As discussed previously, this yields great flexibility in determining when and where to start a given metajob.

Perhaps the most important feature of the Maui scheduler with respect to this research is its ability to run in simulation mode. Given a trace file of an actual workload, the Maui scheduler can simulate the scheduling of the workload, allowing the administrator to experiment with different parameters and attempt to improve scheduler efficiency. The scheduler steps forward in discrete amounts of time. It can be told to single step through these discrete time blocks or to advance through a number of them. Another unique feature valuable to this research is the ability to externally insert new jobs and reservations into the simulated workload as the simulation is running.

3.3 Brigham Young University Meta-scheduler (Ursala)

At Brigham Young University, we have created Ursala, a metascheduling system which creates coinciding advance reservations on participating local schedulers. The metascheduler communicates with each local scheduler, for this research, Maui, using the following steps:

1. Ursala contacts each Maui scheduler requesting resource availability and cost information for a specific job and time frame.
2. Each Maui scheduler incorporates existing reservation, resource, policy, and priority information to determine when resources could be made available for the specified job.
3. Each Maui scheduler reports its finding to Ursala as a list of start time ranges, costs, and resource sets.
4. Ursala receives the lists of start time ranges from each Maui scheduler, computes the intersection of the range lists and determines a best start time and collection of resources.
5. Ursala attempts to create needed reservations for this job on each Maui scheduler. If any reservation attempt fails, Ursala releases all existing reservations for this job, and after a 'backoff' time, returns to step 1. If all reservations succeed, Ursala advances to step 6.
6. With all reservations made, Ursala submits the proper job components to each local resource manager.

The current design implements a reservation release and backoff algorithm to handle the livelock possibility described earlier. Figure 4 is a representation of the basic architecture. Ursala communicates with the local scheduler to obtain resource state information and to create advance reservations. It also interfaces to the local resource management systems for submitting metajob components. Ursala currently supports three scheduling modes described earlier: Specified Co-Allocation, General Co-Allocation, and Optimal Scheduling.

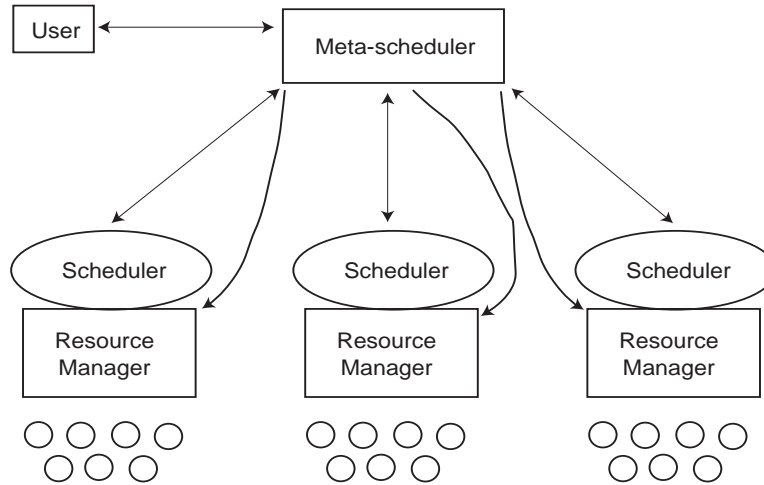


Fig. 4. Meta-scheduler architecture.

4 Experimental Results

The goal of this research is to determine what effects advance reservations have on local scheduling performance and system utilization. To accomplish this, a series of experiments were designed. In this section, we present the experimental environment and results.

In each experiment, the Maui scheduler was used in simulation mode as the local scheduler. Simulation traces were provided to simulate a 192 node IBM SP system with memory sizes ranging from 128 to 512 MB per node. Workload traces representing several weeks of actual workload were used for each simulation run. Each workload trace contains every scheduling aspect relevant to a job, including submit time, submitting user and group, resource requirements, wallclock limit, and actual run time. Using this information, the Maui scheduler is able to introduce the job into the queue at the recorded queue time as the submitting user allowing local scheduler throttling policies to be enforced (e.g. Max Jobs Per User). At that point, the job is scheduled according to the policies and algorithms that currently set for the Maui scheduler. If no changes are made to the configuration of Maui, the simulation of the trace will yield the same resulting schedule as if the jobs were scheduled and run on the real system.

The job traces used for this experiment contained a mix of jobs requesting between 1 and 128 nodes and requiring between 2 minutes and 36 hours to run. The trace represents a period of 2 weeks at the Maui High Performance Computing Center. See [7] for more details about the job trace used. For each experiment, a simulation time period of 10 days was run and analyzed. Since every scheduling-relevant aspect of node resources and jobs is captured in the traces, differences between simulated and actual 'real world' runs are minimal.

To represent the metajobs, a random set of batch jobs were extracted from the simulation job traces. This yields a sampling of actual jobs rather than creating a hypothetical set of jobs for metajob submission. The remaining jobs were then run in the simulation mode of Maui to get baseline utilization and XFactor statistics. A coordinator process was created to read the metajob trace file and advance simulated time for both Ursala and the participating Maui schedulers. At the appropriate times, this coordinator process submitted jobs from the metajob trace file into Ursala's queue. Because Ursala only sees its metajob queue and information returned via its interfaces to the local schedulers, it cannot tell that it is running in a simulated environment and behaves exactly as it would if run in 'production' mode. As the simulation is advanced, statistics from each Maui scheduler are collected and analyzed to determine performance impact of the introduced metascheduled workload.

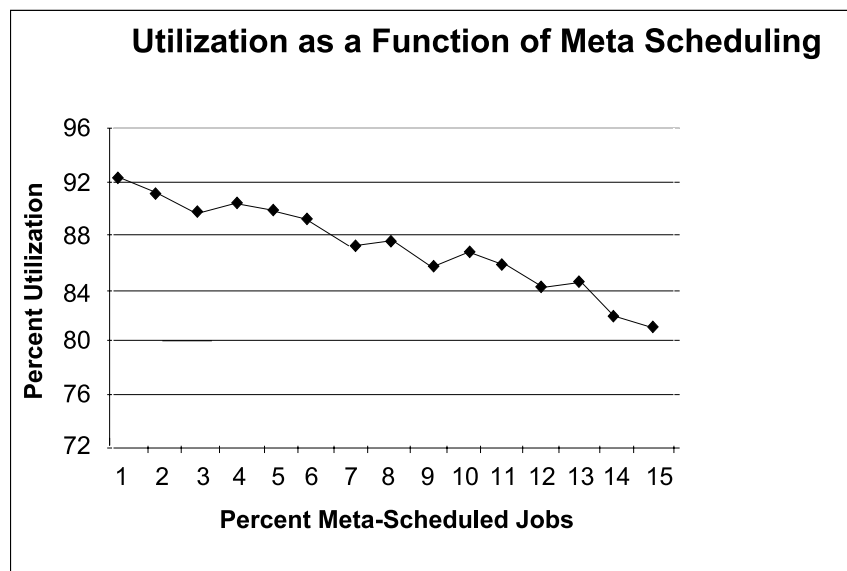


Fig. 5. The effect of advance reservations on system utilization.

The first experiment is a detailed examination of the effects of advance reservations. Ursala was used to insert metajobs requiring advance reservations into the Maui simulation. Initially, no metajobs were inserted, the entire simulation was run to completion, and statistics were gathered. The simulation was then run repeatedly, increasing the percentages of metajobs inserted into the workload mix in each run. Figure 5 shows the resulting system utilization graph for this experiment. Note that, as the percentage of metajobs increases, the overall system utilization declines. This is expected due to the added constraint of a guaranteed start time for each metajob. As a general rule, every added con-

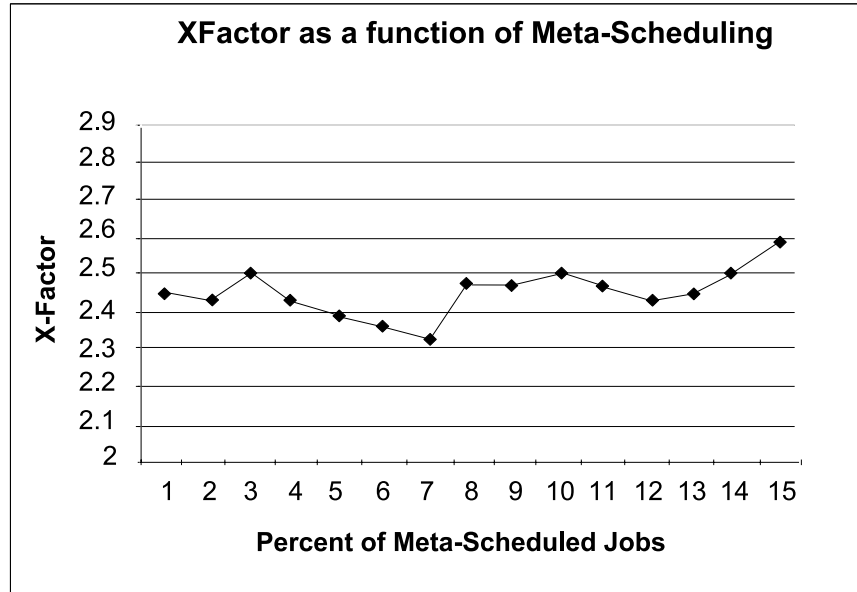


Fig. 6. The effect of advance reservations on expansion factor.

straint will decrease scheduling flexibility and thus decrease resource utilization. This constraint is no exception. Jobs requiring a dedicated start time fragment the scheduler's time space as well as its node space making it more difficult for the scheduler to utilize idle resources. In addition, due to inaccuracies in user's wallclock limit estimations, most jobs will complete early. While normal workload can take advantage of the now available resources by being started earlier than originally planned, metajob components cannot since doing so would cause this component to start before the metajob's other components. Consequently, these resources may go unused.

As previously shown in Figure 1, when advance reservations are added to the workload mix, the scheduler must fit jobs into a two-dimensional mapping. The optimal placement of jobs into this mapping can be reduced to a two-dimensional bin packing problem which is NP-complete[12]. As more advance reservations are added to the mix, placing backfill jobs into the map becomes more difficult. Thus, holes are created and the system utilization goes down. If the accuracy of job run-time estimates were improved, backfill job placement would also be more accurate and somewhat alleviate this effect, but not eliminate it.

The average job expansion factor was also recorded in this experiment, and the resulting graph is shown in Figure 6. The expansion factor is a measure of job turnaround time. While the canonical definition of expansion factor is $(1 + (\text{QueueTime}/\text{WallClockLimit}))$, Maui scheduler uses a modified version of this to incorporate the affect of wallclock limit inaccuracies, namely, $(\text{QueueTime} + \text{RunTime})/\text{WallClockLimit}$. Regardless, the expansion factor calculation scales

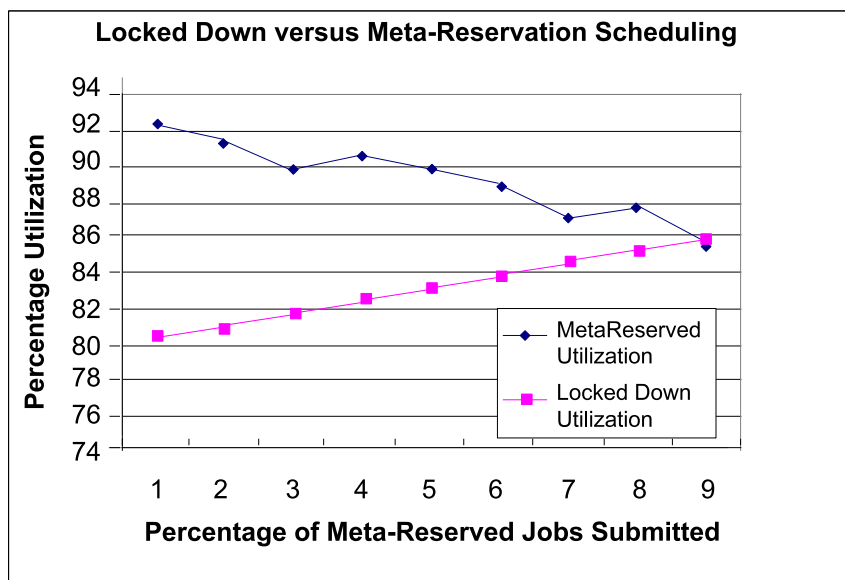


Fig. 7. Comparison of metascheduling based on advance reservations vs. locking down the resources.

the job's actual turnaround time by the job's requested length. Since metajobs increase the number of holes in the two-dimensional node-time map, it becomes harder to backfill. This decreases utilization and increases the average expansion factor, as is reflected in the graph. As the percentage of metascheduled jobs increases, the average expansion factor also increases because QueueTime is increased.

Experiment two is a direct comparison between advance reservation and dedicated resource based strategies for metascheduling. As described before, current 'dedicated resource' metascheduling systems lock down a set of resources to be used by the metascheduler for a specified time frame each day. The findings of this research suggest that scheduling advance reservations for metajobs is more efficient than the blocked dedicated resource approach. The graph in Figure 7 shows the system utilizations of the two approaches. In this experiment, the system in question controlled 192 nodes. For the locked down node metascheduling strategy, 48 of the nodes were blocked out for 8 hours each day for the metascheduled jobs. This was compared to a system using Ursala to schedule advance reservations for the same jobs. The number of metascheduled jobs increased in each simulation such that 1% metascheduled jobs corresponds roughly to 10% utilization of the dedicated nodes. Since we are using 25% of the nodes for 1/3 of each day, 8.3% metascheduled load would be roughly 100% utilization of the locked down nodes. With a small percentage of metajobs, advance reservation metascheduling yields much higher utilization. Larger percentages of metasched-

uled jobs bring the two curves closer together, and finally, the dedicated node strategy will result in slightly higher overall utilization. However, it should be noted that the utilization assigned to the block of dedicated metascheduler resources was assigned arbitrarily in this experiment, i.e. the utilization on the blocked nodes was increased rather than attempting to schedule an increased metascheduling workload onto these resources. This removed the issues of fragmentation and packing metascheduled workload. In reality, since both the local workload and the metascheduled workload contained a similar mix of jobs, but the metascheduled resources were relatively smaller, the metascheduler would be able to obtain a system utilization which approached but did not exceed that obtained by the local scheduler. Also, since average system utilization is generally proportional to (average job size) / (total resources available), neither the local scheduler nor the metascheduler would be able to obtain the level of utilization obtained when the resources were not fragmented.

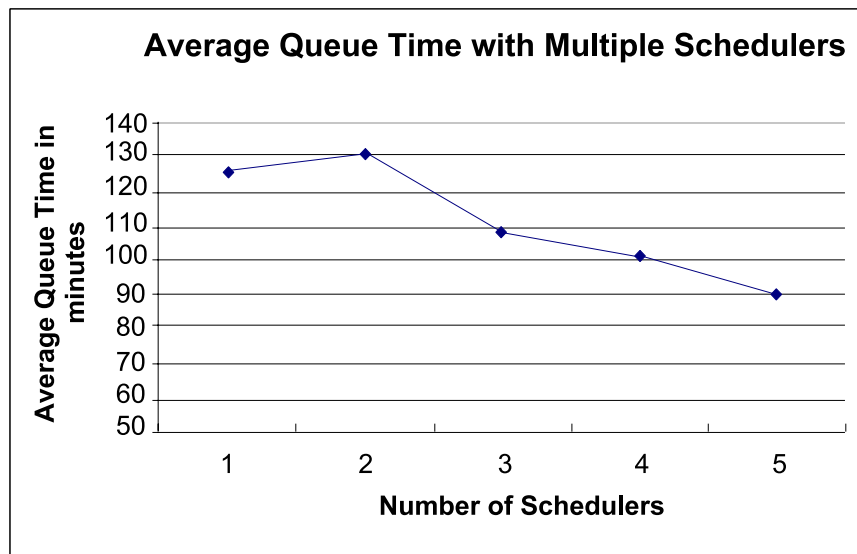


Fig. 8. Meta-scheduled job queue time comparison using a single local scheduler vs. using multiple local schedulers.

The final experiment is set up to show the benefits of metascheduling using the resources at multiple sites. In this case, Ursala was connected to multiple Maui schedulers; each running a different simulation trace file. Ursala processed a job trace using a simple 'ASAP' utility function. Each metajob in this experiment only required resources at a single site. This allowed Ursala to minimize queue time for each metajob and showcase the benefits of such an approach.

The graph in Figure 8 shows the average queue time for the metajobs as more local systems are added to the simulated metacomputer. It is clear that as more systems are added, the probability of finding an earlier start time for a metascheduled job increases even though each local system maintains a high local system utilization. This is due to the fact that there is a greater probability of finding an appropriate sized hole for the metascheduled job earlier on one of the local systems. Thus the queue time for the metajobs decreases.

5 Conclusion

An advance reservation based metascheduling system provides significant advantages over a metascheduler based on blocked resources dedicated to metajob usage. Such a system allows metajobs to run at any time and not be limited to dedicated resource blocks and time frames. The 'integrated' approach of allowing the local scheduler to determine when and which resources are available for metajobs, allows the local scheduler opportunities to fully enforce local policies and to fully optimize scheduling and resource allocation. This results both in more local control over local resources and also better utilization of these resources. These advantages make it likely that a reservation based metascheduling system will become the standard in the future. The architecture outlined in this paper promises to minimize the negative impact of a metascheduled workload. However, due to past experience, there is still resistance to any metascheduling system at many sites. This paper focused on the performance impacts on local workload of the reservation based metascheduler. This impact was quantified across a range of loads and demonstrated to be much lower than that found under a blocked resource metascheduler. This information will allow local management and administrators to make informed decisions regarding whether or not the benefits of the metascheduling system justifies the cost. The research here also yields the information needed for intelligent setting of metajob throttling policies which would bring this impact to a tolerable level.

The Grid Forum[13] has recently been formed to examine issues related to establishing a nationwide computational power grid. This group is attempting to create an architecture that will allow users to submit jobs to the grid without needing to know the exact location where their jobs will run. Advance reservations are an important component of the overall grid system. The infrastructure created for this research will enable investigators to answer questions about several design decisions that must be made in creating the grid. This paper indicates how varying metascheduled workloads affect the quality of service delivered to local jobs thus allowing scheduler administrators and management to determine acceptable impacts and set appropriate policies to throttle such external workload.

Future research will compare stateless reservation creation vs. courtesy reservations. Additional research will also be conducted in the area of 'Optimal' job scheduling and in ways of utilizing current performance information in deciding optimal metajob fragmentation and resource allocation. Answering these ques-

tions in a quantitative way is important as a step towards validating the feasibility of a wide-spread metascheduling environment. Such research is vital in the effort to persuade local system administrators to make their systems available for use by metajobs and demonstrate that an advance reservation metascheduler will increase the utility and availability of computational resources in the supercomputing and cluster community.

References

1. P. Chandra *et. al.* Darwin: Resource management for value-added customizable network service. In *Sixth IEEE International Conference on Network Protocols*, 1998.
2. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
3. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *IJSA*, 11(2):115–128, 1997.
4. Andrew S. Grimshaw and William A. Wulf. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), 1997.
5. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *International Workshop on Quality of Service*, 1999.
6. Maui High Performance Computing Center. The maui scheduler. In <http://www.mhpcc.edu/maui/>, 1999.
7. Mark Clement, Quinn Snell, David Jackson, and David Ashton. High performance scheduling for windows nt. In *Proceedings of the 1999 International Conference on Parallel and Distributed Techniques and Applications*, pages 525–531, 1999.
8. Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35:45–52, June 1992.
9. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.
10. Abraham Silberschatz and Peter Baer Galvin. *Operating System Concepts*. Addison-Wesley, 1998.
11. R. Metcalf and D. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–403, 1976.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness*. W.H. Freeman, 1979.
13. The Grid Forum. The grid forum. In <http://www.gridforum.org/>, 1999.