

Performance Impact of Resource Provisioning on Workflows

Gurmeet Singh, Carl Kesselman and Ewa Deelman
Department of Computer Science
University of Southern California, Los Angeles, CA 90089

Abstract

Resource provisioning refers to the dedicated use of certain set of resources for executing an application. In this paper, the application workflow is a set of tasks, represented as a directed acyclic graph (DAG). Provisioning can be done statically using advance reservations or using dynamic provisioning mechanisms such as Condor Glidein. A simulation is done using the Maui simulator, workload trace collected from the NCSA Teragrid cluster and 25 workflows to study the effect of provisioning on the completion time of the workflows and the utilization of the provisioned resources. The simulated results show in general a reduction of about 50% in the workflow completion time for the workflows using the FIFO and the Fair share scheduling policies. In some cases, an order to magnitude reduction in completion time is obtained. We also address the issue of utilization of the provisioned resources using a simple metric which tries to maximize the ratio of utilization to the completion time. Finally we present the results of a survey on the support of advance reservation at the Grid sites.

1. Introduction

Computational Grids provide access to software and hardware resources that are geographically distributed and belong to different administrative domains. The availability of storage and compute resources at the Grid sites facilitate execution of applications with large resource requirements. Several programming paradigms are commonly used in developing applications on distributed systems e.g. Master-Worker, Single Program Multiple Data (SPMD), Data Pipelining, Divide and Conquer, and Speculative Parallelism [1]. In this paper, we use a workflow model where the tasks have fixed and specified dependencies. The task graph is in the form of a directed acyclic graph (DAG) and is known as the application workflow [2].

The Grid infrastructure is typically organized as a collection of sites, where each site has a collection of compute and storage resources and is administered independently of the other sites. Each site has a local resource management system that is responsible for allocating the available resources to the submitted jobs according to the specified policies. The resources at each Grid site are shared by both local and remote users. If the resource requirements of a submitted job cannot be met immediately, it is put into a queue of waiting jobs. Minimizing the execution time of an application in such a shared environment is a challenging problem. The completion time of an application depends not only on the resource requirements of the application, but also on the current and future workload and the scheduling policies of the sites where it is scheduled.

In this paper, we study the impact of using resource provisioning on the workflow performance. Using resource provisioning, dedicated resources are made available for executing a workflow. With the provisioned resources, the workflow completion time can

be determined by analyzing the workflow structure and considering the amount of provisioned resources. Thus, it brings more predictability to the workflow performance. However, there is a cost associated with provisioning the resources. In this paper, it is modeled as the waiting period before the resource can be obtained. The performance of the workflow is compared when it is executed without provisioning versus when it is executed with provisioning. The performance metrics that we are interested in are the workflow completion time and the utilization of the provisioned resources.

Resource provisioning can be done statically using advance reservations or dynamically using mechanisms such as Condor Glidein [3]. Using advance reservations a scheduler agrees to make certain resources available for certain timeframe for a specified user or set of users. All the jobs submitted by the user during this timeframe are scheduled on the reserved resources. There is generally no incentive for making reservations for a single job since policy decisions will likely be made so that there is no start time advantage to making a reservation over submitting the job to a queue [4, 5]. However for a workflow, all the tasks in the workflow can be executed using a single resource reservation, eliminating the queue wait time and so the completion time is expected to be reduced. A notice period has to be given before the reservation can become active. The scheduler uses this notice period to make sure that no currently queued jobs would get unnecessarily delayed because of this reservation. In other words, the scheduler tries to make sure that no start time advantage is obtained by making advance reservation instead of submitting the job to the queue. Resource provisioning can also be done dynamically using mechanisms that can provide temporary access to dedicated resources from a remote site without requiring extra coordination with the remote scheduler. An example of dynamic resource provisioning is Condor Glidein [3]. It submits a job to a remote site. When this job starts executing on the allocated resources, it starts up processes on them that allow job submission to these resources bypassing the remote scheduler. Thus it created a dedicated execution environment on the set of allocated resources controlled by a scheduler local to the user. The cost associated with dynamic provisioning for Condor Glidein is the wait time before this main job can start running.

When using resource provisioning, an important decision to be made is how many resources should be provisioned and for how long. The delay incurred in obtaining the provisioning often depends on the amount of resources requested. The resources provisioned should be enough to execute the workflow to completion. A number of provisioning requests can be determined differing in the amount and duration of resources requested that are sufficient to execute a particular workflow. A particular request has to be selected among these that optimize the metric of interest. Note that it may not be possible to determine exactly which request will do the best because the obtained performance not only depends on the current workload and the scheduler policies but also the future workload which cannot be reasonably known in advance. Request selection for improving the turnaround time of parallel jobs has been studied extensively in the past [6-8]. We extend it for workflows and also take the utilization of the requested resources into consideration.

We use the Maui Simulator [9] with a workload trace obtained from the NCSA Teragrid cluster for the evaluation. We use 25 workflow graphs, three scheduling policies (FIFO, a variant of fair share with and without backfilling) and three provisioning models

(no provisioning, dynamic provisioning and advance reservation) for our study. The results indicate that depending on the scheduling policy and the workload, resource provisioning can greatly improve the workflow completion time. In general, a 50% reduction in the workflow completion time was obtained when using the FIFO and the fair share scheduling policies. With the latter policy, an order of magnitude reduction was obtained for some workflows. The amount of reduction in the completion time seems to depend on the size of the workflow in terms of number of tasks in it. However, with the backfilling scheduling policy we did not see any performance benefit except for very large workflows. We also show that the utilization of the provisioned resources can be increased at a small expense of the workflow completion time.

The rest of the paper is as follows. Section two describes the simulation that we have used for evaluating the performance impact. Section three describes the request selection policy and the mechanisms used for simulating advance reservations and dynamic provisioning. The workflow graphs are described in Section four. Section five presents the experimental results where the completion time is the metric of interest. Issues of resource utilization are addressed in Section six followed by a discussion in section seven. Section eight presents the results of a survey on the support for advance reservation at the Grid sites. Related work is presented in section nine and conclusions and future directions are presented in section ten.

2. Simulation

A workload trace was collected from February 22nd to March 3rd, 2005 by querying the scheduler periodically at the NCSA Teragrid cluster facility. The trace contains 2911 jobs. Out of the 2911 jobs submitted, only 2095 jobs actually ran and were considered for the simulation purposes. The trace is formatted so that it could be understood by the Maui simulator. The simulated cluster had 890 processors. The simulator is initialized with a set of 102 jobs that were running when the trace collection started. Then a simulator controller advances the simulation time and submits jobs to the simulator according to the time when they were submitted in the workload trace. The simulation controller is also responsible for parsing the workflow description and submitting jobs from the workflow to the simulator as they become ready as shown in Figure 1.

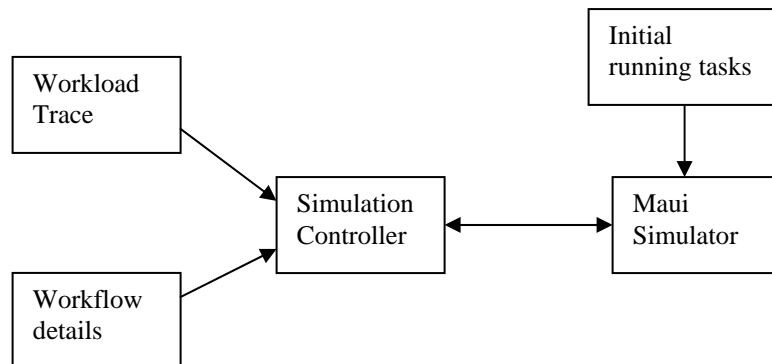


Figure 1: Workflow and Workload simulation.

The controller can limit the number of tasks in the workflow that can be submitted to the simulator at any given time. This is intended to model a real world scenario where the number of jobs that are submitted to a remote resource is limited due to the middleware

costs involved, resource limitations at the central points of job submission, local policies that limit the number of running tasks of any particular user and the desire not to overload and annoy the system administrators of the supercomputing or Grid sites. For the purposes of this paper, we set the limit to 100 submitted tasks at any given time unless otherwise mentioned.

We simulated three scheduling policies in the Maui simulator. The first policy was FIFO (First-in, First-out), the second policy is a variant fair share that gives priority to jobs according to the following criteria

- Fair share on user based historical usage
- Service (combination of queue time incurred, expansion factor and the number of times the job has been bypassed by lower priority jobs). The expansion factor is defined as $(1 + \text{queuetime}/\text{wallclocktime})$. It is intended to give priority to short jobs.
- Resources. It gives priority to jobs based on the number of resources requested thus giving higher priority to large job in order to ensure they don't starve. The supercomputing centers often have a mandate to run large parallel jobs that cannot run elsewhere.

We call this policy as fair share for convenience in the rest of this paper. The third policy is the same as the second policy but it also incorporates backfilling. We use conservative backfilling where the 50 highest priority jobs in the queue are not affected by the backfilling. The backfilling algorithm used is FIRSTFIT. For the Maui scheduler, FIRSTFIT selects the first job that will fit in the available resource space [9].

The job description submitted to the simulator by the controller contain the number of processors requested, the requested running time and the actual running time of the task as determined by analyzing the workload trace.

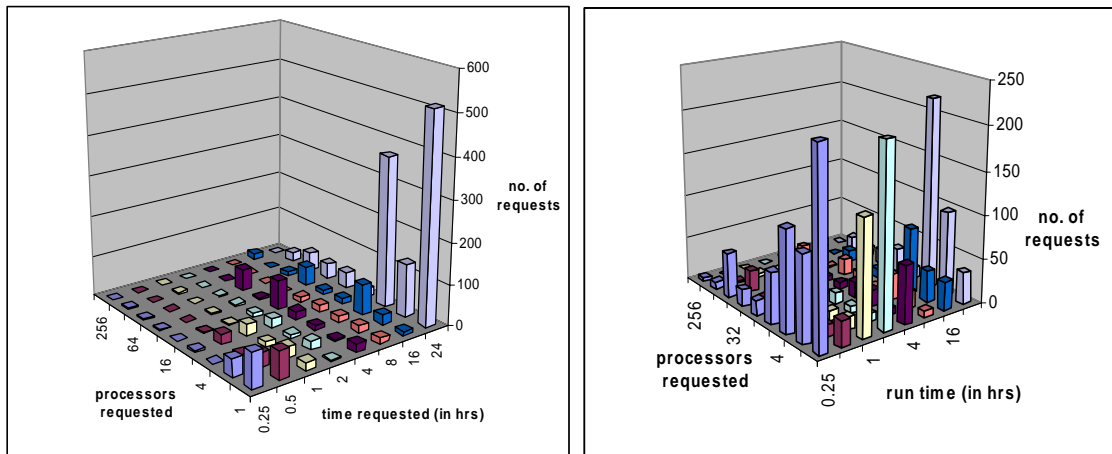


Figure 2: requested time (left) and actual run time (right) distribution of the jobs that finally ran on the cluster.

Figure 2 shows the distribution of the time requested by the jobs and their actual runtimes generated by analyzing the workload trace. The figure also shows the distribution of the number of processors requested. 55% of the submitted jobs request a runtime between 16 and 24 hours whereas only 21% of the jobs run that long. This might be because the users are not sure of the runtime of their jobs and hence request the maximum allowed duration.

3. Resource Provisioning

This section describes the mechanisms used for simulating advance reservation and dynamic provisioning. It describes the request selection criteria for selecting among multiple possible requests.

3.1 Advance Reservation

Advance reservation implies a guarantee by the remote scheduler to make certain resources available for a user or set of users for certain timeframe. In order to make a reservation, the user sends a request to the scheduler containing the number of resources requested and the duration. The scheduler replies with an earliest start time for the reservation after making sure that no currently queued job would get delayed and no currently running job would need to be terminated in order to provide this reservation. Note that the scheduler makes this decision based on only the requested times of the running and queued jobs. It cannot take the actual runtime of the jobs into consideration since it is not known.

In order to estimate the earliest start time that the scheduler would reply to a reservation request, we do a simulation run with the Maui simulator where we make the actual run times of the jobs in the workload equal to their requested times and submit a job whose resource requirement is similar to what we want to reserve. No job is submitted after this job since future requests cannot affect the decision. The start time of the job with this simulation signifies the earliest start time, the reservation can be made taking the current scheduling policy into account.

3.2 Request Selection

There may be several resource request that can execute the workflow. Figure 3 shows a simple workflow with 5 tasks, each of which take 5 minutes to complete. The figure also shows three requests (no. of processors and requested time) that are sufficient to execute the workflow. The data transfer between the tasks is done using files in a shared file system and the file access time is included in the task execution time.

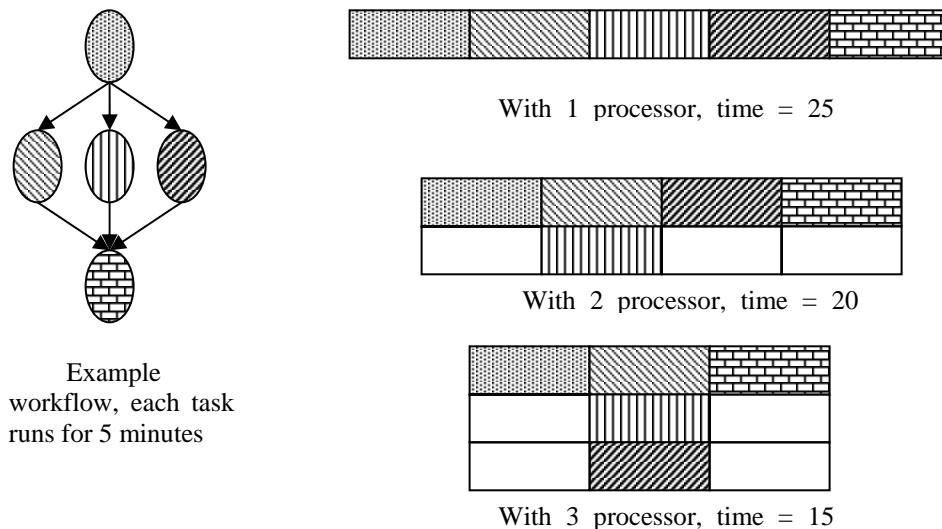


Figure 3: A workflow and three possible resource requests.

Since a workflow can be executed using a number of requests, the selection of the request that is finally used depends on the earliest start time of the request as determined by querying the scheduler and the metric to be optimized. When the completion time is the metric of interest, the request is selected that minimizes the sum of the wait time and the execution time of the workflow on the requested resources. The wait time is the difference between the time when the scheduler is queried and the earliest start time of the request as determined by the scheduler. In order to save simulation time, we do not actually simulate the execution of the workflow over the reserved resource but compute the execution time of a workflow on the provisioned processors using a variation of the Earliest Time First algorithm [10] as shown below.

- (1) Determine the parent and child tasks of each task in the workflow.
 - (2) Initially, the ready list only contains the tasks with no parents.
- Repeat
- (3) Take the first task from the pool of ready tasks.
 - (4) Find the earliest possible start time of the task based on the completion time of the parent tasks.
 - (5) Find the earliest possible start time on each processor subject to the constraint in (4)
 - (6) Schedule the task on the processor that gives the earliest start time. Update the task completion time and the processor availability time. Mark the task as completed.
 - (7) Search the child tasks of this task and if they are ready, add them to the ready list. Remove the current task from the ready list.
- Until the ready list is empty
- (8) The last availability time among the set of processors gives the workflow completion time.

The time complexity of the above algorithm is $O(n^3)$ where n is the number of tasks in the workflow and $n > p$, the number of processors in the set. The implementation of the algorithm considers the fact that the scheduler only makes scheduling decisions at periodic intervals.

For determining the minimum completion time of a workflow using advance reservation, we determine a number of possible requests for the workflow. The earliest start time of the requests using advance reservation is determined by simulation as described in the previous section. The request that minimizes the sum of the wait time and the execution time determines the minimum completion time of the workflow using advance reservation.

3.3 Dynamic Provisioning

For estimating the minimum completion time with dynamic provisioning, we determine the request that will perform the best with advance reservation. Another simulation is done with the actual run time of the jobs as recorded in the trace and this request is submitted as a job in the simulation. The time when this job starts running gives the actual start time of the job. The completion time of the workflow using dynamic provisioning is given as

Completion time = (actual start time of job – submit time of job) + execution time.

The simulation is stopped when the job starts running and the execution time of the workflow is determined using the algorithm in the previous section. The actual start time

of the job can be different from the earliest start time estimated by the scheduler in advance reservation because of two reasons. Firstly, the scheduler uses the requested wall clock times of the submitted jobs for its estimation. The actual runtime of the submitted jobs can be much less than the requested wall clock time as shown in Figure 2 for the workload trace used. Secondly the actual start time of the job can be affected by jobs that are submitted later (unless the scheduling policy is FIFO), something that cannot be taken into consideration by the scheduler when estimating the earliest start time. The actual start time of the job was almost always less than the estimated earliest start time in the experiments in this paper.

4. Workflow graphs

We use 25 workflow graphs for doing our evaluation. All of them have the same sequential runtime. One of them is a workflow from an astronomical application called Montage [11]. The rest of the workflow graphs are created by applying transformations to the Montage workflow that have the effect of changing the structure and the granularity of the Montage workflow. The completion times of the workflows described in this section would be used to evaluate the performance impact of resource provisioning. All the workflow graphs are described by their average parallelism, the ratio of maximum to average parallelism, total number of the tasks and the average runtime of tasks in the workflow. The average parallelism as defined in [12] as the ratio of the total runtime of the graph to the length of the longest path in the graph.

4.1 Montage workflow

Montage is an application for constructing astronomical image mosaics on demand. The structure of a small Montage workflow is shown in Figure 4. The vertices in the graph represent the compute tasks and the directed edges represent the data dependencies between the tasks.

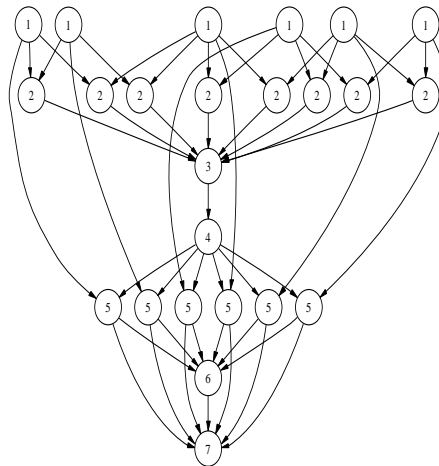


Figure 4: A small montage workflow.

For ease of reference, we assign a level to each job in the workflow indicated by the number inside the vertices in Figure 4 using the following algorithm from [13].

```

for all tasks which do not have any predecessor
    level = 1;

```

```

for each task(i) which already has a level assigned to it
{
  for all its unassigned output tasks
    level = level of task(i) + 1;
  for all its assigned output tasks
    if level of output task is less than or equal to the level of task(i)
      level = level of task(i) + 1;
}

```

The jobs at the same level are independent of each other. All the graphs used in this paper have 7 levels. The workflow that we have selected is used to create a 10 square degree mosaic of the M16 region of the sky. The workflow contains 17034 tasks and the average runtime is 3.1 seconds. The average parallelism is 89 and the ratio of maximum to average parallelism is 113.

For generating the other workflow graphs, we cluster the tasks in the Montage workflow. This technique has several advantages over generating random workflow graphs. Firstly it gives us executable graphs of a real application with different granularities and structures to work with. Secondly the sequential execution time of all the graphs is the same since they are the transformations of a single graph, thus providing a solid base for comparing the results. Lastly, as an aside, the results suggest the best technique for executing the Montage workflow.

4.2 Fixed number of clusters per level

In this type of clustering, we cluster the tasks at each level in the workflow into a fixed number of clusters.

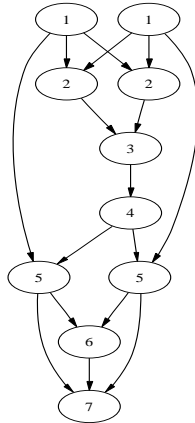


Figure 5: Montage workflow with 2 clusters per level.

Figure 5 is the resulting graph when the Montage workflow in Figure 4 is clustered with 2 clusters per level. The number of clusters per level is the clustering parameter. The 10 degree Montage workflow is clustered with 1, 2, 4, 8, 16, 32, 48, 56, 64, 128, 256, 512 clusters per level, thus generating 12 different workflow graphs.

Table 1: Workflow characteristics with fixed clusters per level.

Clustering parameter	1	2	4	8	16	32	48	56	64	128	256	512
Average parallelism	1	1.97	3.87	7.42	13.68	23.69	31.22	34.66	38.36	54.23	67.2	77.34
Ratio of max to av	1	1.01	1.03	1.07	1.16	1.35	1.53	1.61	1.66	2.36	3.8	6.6

parallelism												
Total no. of tasks	7	12	22	42	82	162	242	282	322	606	990	1758
Average runtime (seconds)	7546	4402	2401	1257	644	326	218	187	164	87	53	30

Table 1 gives the characteristics of the workflow graphs generated by this method. For these graphs, the clustering parameter denotes the maximum parallelism of the graph.

4.3 Fixed number of tasks per cluster

The previous clustering resulted in fixed number of clusters per level. But the number of tasks grouped in each cluster differed between the levels because the original Montage workflow has different number of tasks at each level. In this section, we group the same number of tasks in each cluster across levels.

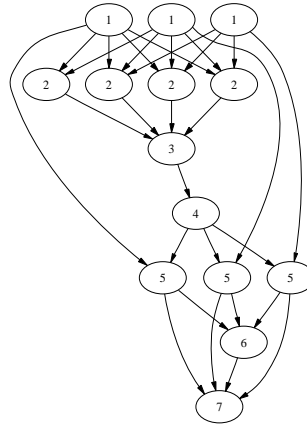


Figure 6: Montage workflow with 2 tasks per cluster.

Figure 6 shows the resulting graph when the Montage workflow in Figure 4 is clustered with 2 tasks per cluster. The number of tasks per cluster is the clustering parameter. The 10 degree Montage workflow is clustered with 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 tasks per cluster resulting in 12 different workflow graphs. These graphs are similar in structure to the original Montage workflow with coarser granularity tasks. The original Montage workflow is a special case of this clustering when the clustering parameter is one.

Table 2: Workflow characteristics with fixed number of tasks per level.

Clustering parameter	2	4	8	16	32	64	128	256	512	1024	2048	4096
Average parallelism	79.2	67.1	54.5	40	26.3	15.9	9.69	7.73	5.47	3.46	2.01	1.25
Ratio of max to av parallelism	63.6	37.5	23.1	15.7	11.9	9.9	8.14	5.17	3.65	2.88	2.48	2.39
Total no. of tasks	8518	4261	2132	1068	537	270	137	72	38	22	13	9
Average runtime (seconds)	6.2	12.39	24	49	98	195	385	733	1390	2401	4063	5869

Table 2 shows the characteristics of the workflow graphs generated. The ratio of maximum to average parallelism is typically more for these graphs than the graphs generated in the previous section.

5. Experiments

This section lists the experimental results obtained by doing the simulations. The goal is to minimize the completion time of the workflow. The first subsection shows the performance of various requests possible for the original Montage workflow in order to show the request selection mechanism. The rest of the subsection show the performance of the best possible request. In all the experiments described in this paper, we never provisioned more than 100 resources since we had put a limitation of a maximum of 100 submitted tasks when executing the workflow without provisioning.

5.1 Montage workflow

The 10 degree Montage workflow takes 3930, 2611 and 794 minutes to complete using FIFO, fair share and fair share with backfilling respectively. The completion time with backfilling is considerably less due the fine granularity of the Montage workflow.

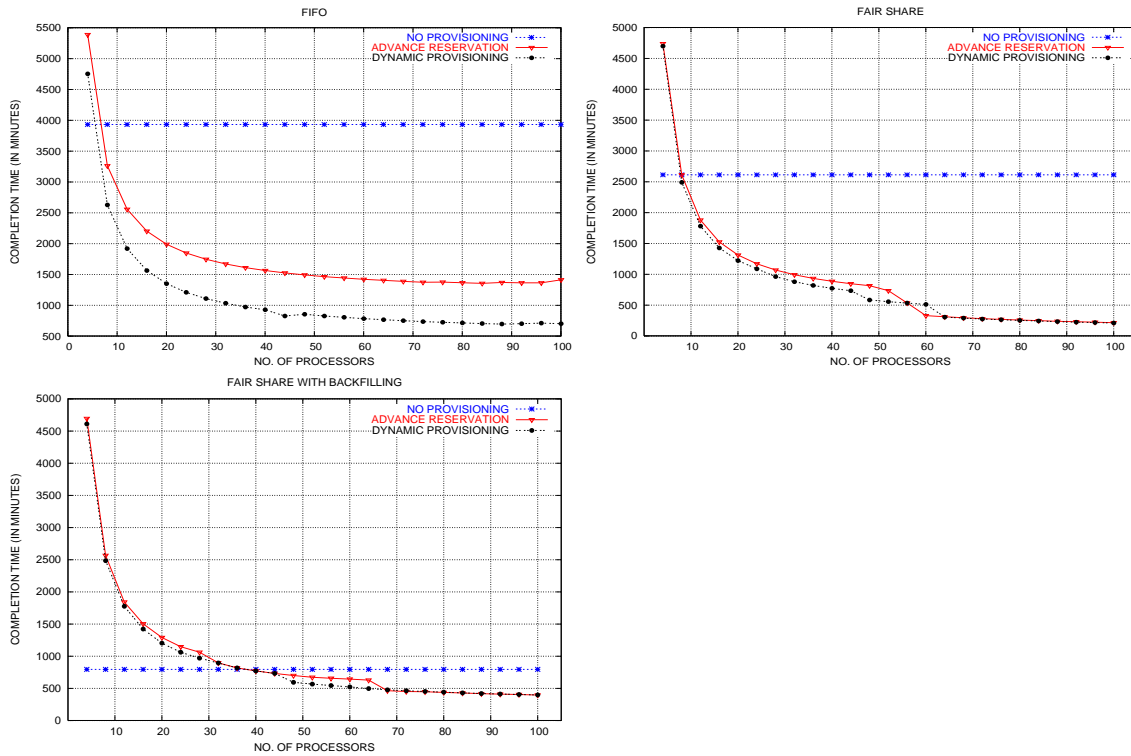


Figure 7: Time taken to complete original Montage workflow with various scheduling and provisioning policies.

Figure 7 shows the completion time of the various provisioning requests that were possible in order to execute the workflow. The x-axis shows the number of requested processors in each provisioning request. The y-axis shows the workflow completion time for advance reservation and dynamic provisioning. For FIFO, the completion time with dynamic provisioning is always less than the advance reservation time because the actual start time of the request is always earlier than the estimated start time. For the other

scheduling algorithms, it is generally less but not always due to the possibility of out-of-order execution. The requests that perform well with advance reservation with respect to the completion time also perform well under dynamic provisioning.

The minimum completion time for FIFO with advance reservation is 1351 minutes (65% reduction over the no provisioning time) when requesting 84 processors for 214 minutes. The same request finishes in 699 minutes with dynamic provisioning (82% reduction). The minimum completion time with dynamic provisioning is 690 minutes with 88 processors. For the fair share and the backfilling scheduling schemes, the workflow completion times are pretty close for the advance reservation and the dynamic provisioning schemes indicating that the estimated start time of the request was a good prediction for the actual start time of the request. The reason for this is that in FIFO, the advantage of earlier jobs in the queue finishing before their requested time accumulate whereas it is not so for the other two strategies due to the possibility of out-of-order execution. The rest of the figures in the following subsections only show the performance of the best request as estimated by advance reservation scheme and its actual performance as observed using dynamic provisioning. We do not pursue the best request for dynamic provisioning as it is not possible to know in advance which request will perform the best.

5.2 Fixed number of clusters per level

Figure 8 shows the completion time for the workflows described in Section 4.2.

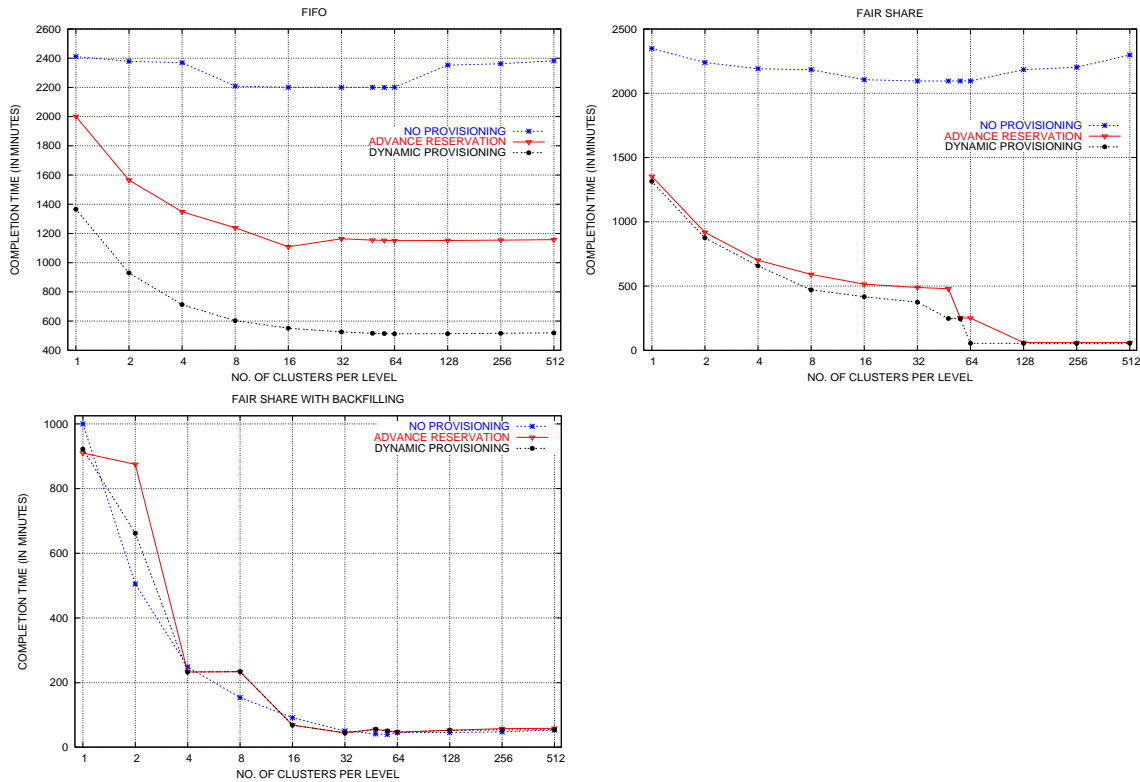


Figure 8: Completion time for workflows with fixed clusters per level with various scheduling and provisioning policies.

As can be seen from the figures, provisioning leads to a significant reduction in the workflow completion time for both the FIFO and the fair share policies. When backfilling

is being used, the completion time with provisioning is more for smaller workflows (<100 tasks). This is because the tasks in the workflow are able to take advantage of backfilling but the provisioning requests, due to their bigger resource requirements are not able to do so.

5.3 Fixed number of tasks per cluster

Figure 9 shows the impact on the workflow completion time with provisioning for the workflow graphs described in Section 4.3.

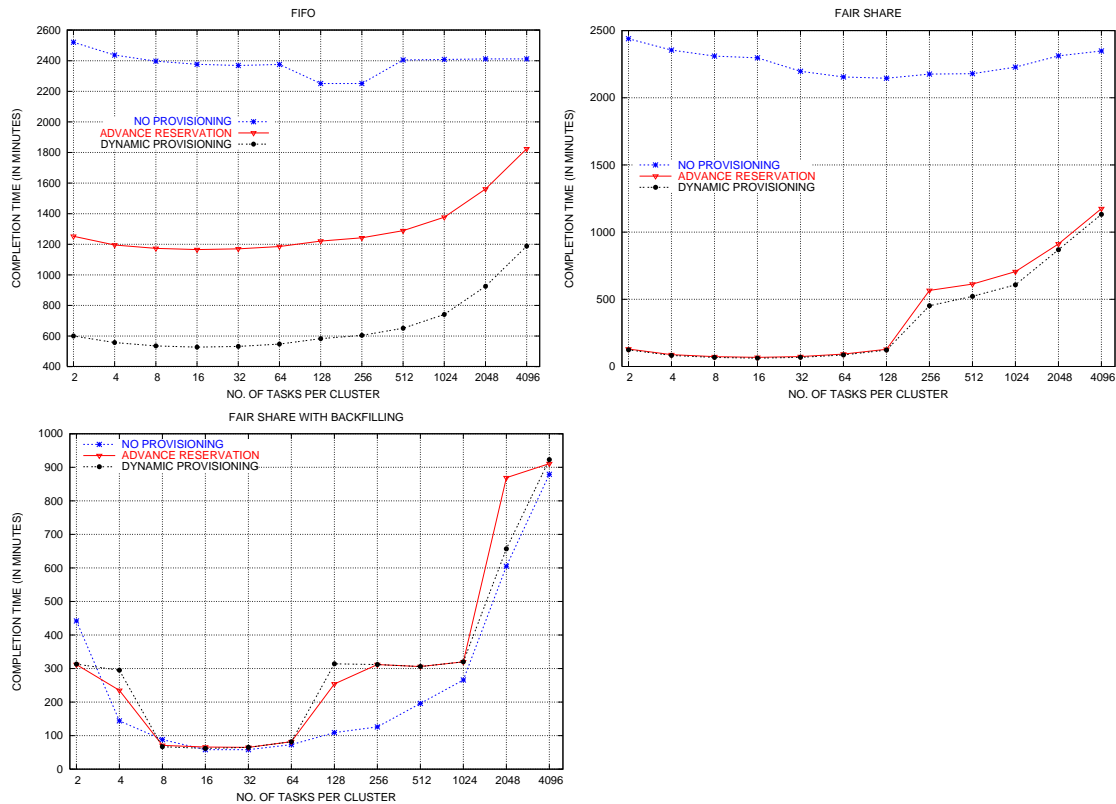


Figure 9: Completion times for workflow with fixed no. of tasks per cluster with various scheduling and provisioning policies.

Again, the results are similar to what were obtained for the workflow graphs with fixed number of clusters per level. For the fair share scheduling policy, an order of magnitude reduction in completion time was possible using provisioning.

6. Resource utilization

When we provision a number of resources for a workflow, it is possible that not all of them would be used all the time. This is due to the sequentiality imposed by the dependencies in the workflow. Figure 3 shows three possible schedules for executing a workflow with one, two and three processors. The utilization of resources in the resulting schedule is 100%, 62.5%, and 55% respectively. Thus there is a tradeoff between the execution time of the workflow on the provisioned resources and the resource utilization. However, maximizing the utilization alone is not very useful, as it is trivially maximized when using a single processor. Thus the metric that we are interested in maximizing is

$$\frac{(\text{utilization})^x}{(\text{completion time})}$$

Where x is a non negative integer, indicating the emphasis we want to put on utilization. $x = 0$ refers to the case when we only want to minimize the completion time. For a particular resource request, utilization refers to the utilization of the resources in the resulting schedule of the workflow on the requested resources and the completion time is the sum of the wait time of the request and the execution time of the workflow on the provisioned resources. Thus this metric also considers the wait time of the requests for the provisioning scheme used. When the workflow is executed without provisioning, it can be considered to operate at 100% utilization since each task requests a resource only for the time when it will be actually running on it. In the following subsections, we will compute the value of the metric with various values of x (0,1,2,3) for each request and show the performance of the resulting request that performs the best for each metric with respect to the completion time and the resource utilization. In the case of the original Montage workflow, when FIFO is the scheduling policy, $x = 0$ and $x = 1$ selects the request with 84 processors while $x = 2$ and $x = 3$ selects the request with 72 processors. The utilization increases from 94.8% to 95.82% while the completion time increases by 18 minutes. The rest of the scheduling policies select the same request for the original Montage workflow for all considered values of x .

6.1 Fixed number of clusters per level

We use the metric described above with various values of x and select the request that maximizes the value of the metric for each value of x . Figure 10 shows the utilization and completion time of each request selected for the graphs described in section 4.2 with the FIFO scheduling policy and using advance reservation.

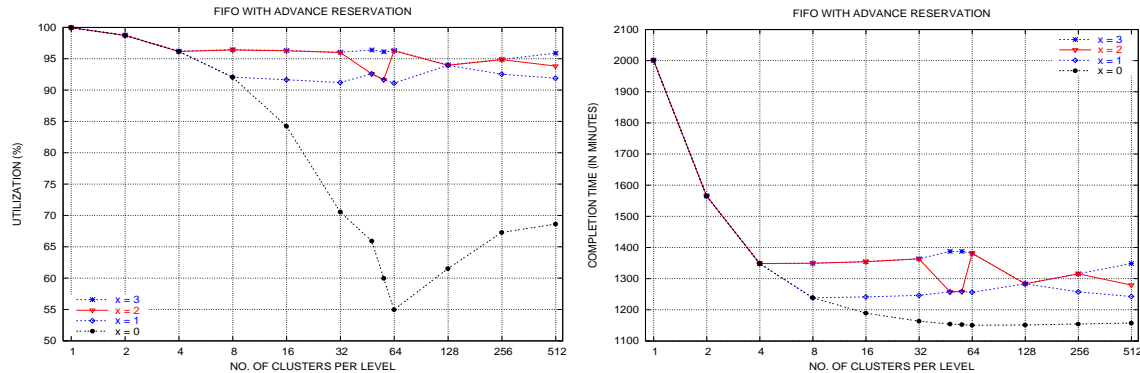


Figure 10: The resource utilization (left) and completion time (right) when requests are selected based on the value of the metric with $x = 0$ to $x = 3$ with FIFO and advance reservation policy.

Figure 10 shows that the resource utilization can be increased by using higher values of x with small increase in the resulting completion time. The reduction in the completion time is still significant considering that without provisioning these workflows take at least 2200 minutes to complete (Figure 8).

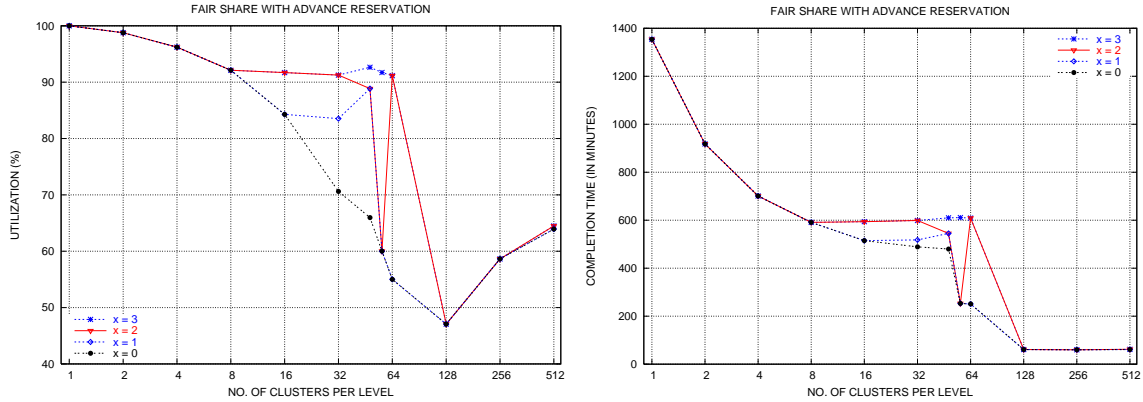


Figure 11: The resource utilization (left) and completion time (right) when requests are selected based on the value of the metric with $x = 0$ to $x = 3$ with Fair share and advance reservation policy.

Figure 11 shows the results for the fair share scheduling policy with advance reservations. For many of these graphs, the request that minimizes the completion time also optimizes the other metrics. This is because of the large disparity in completion time between the request that minimizes the completion time and other requests. Hence the other metrics also choose this request even though the utilization may be less than 65%. Due to space constraints we do not show the results with other scheduling and provisioning policies.

6.2 Fixed number of tasks per cluster

This section shows the results for the graphs described in section 4.3. Figure 12 shows the performance of the various requests selected based on the metrics for the FIFO scheduling policy with advance reservation as in the previous section.

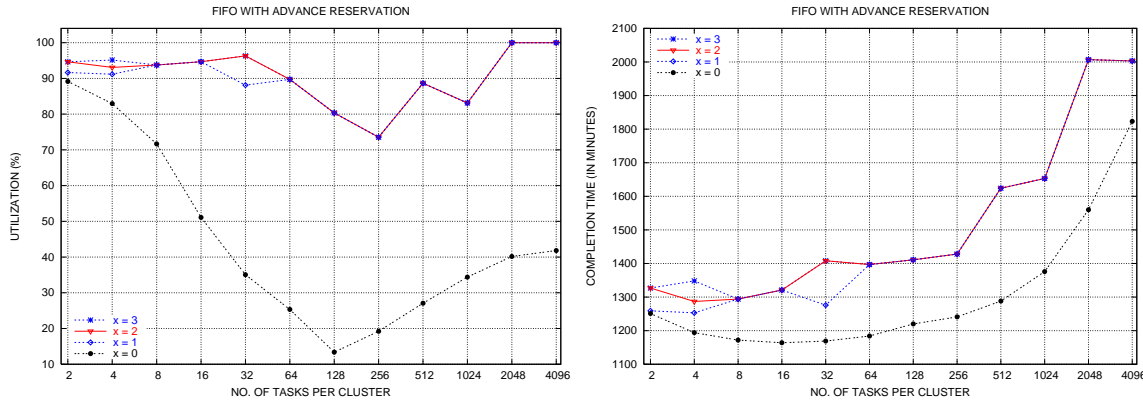


Figure 12: The resource utilization (left) and completion time (right) when requests are selected based on the value of the metric with $x = 0$ to $x = 3$ with FIFO and advance reservation policy.

In general for these type of graphs where the parallelism varies more between the levels, the resource utilization can be less than 20% if we want to minimize the completion time only. As Figure 12 shows, even using $x = 1$ improves the utilization significantly while slightly increasing the completion time that is still less than the completion time with no provisioning.

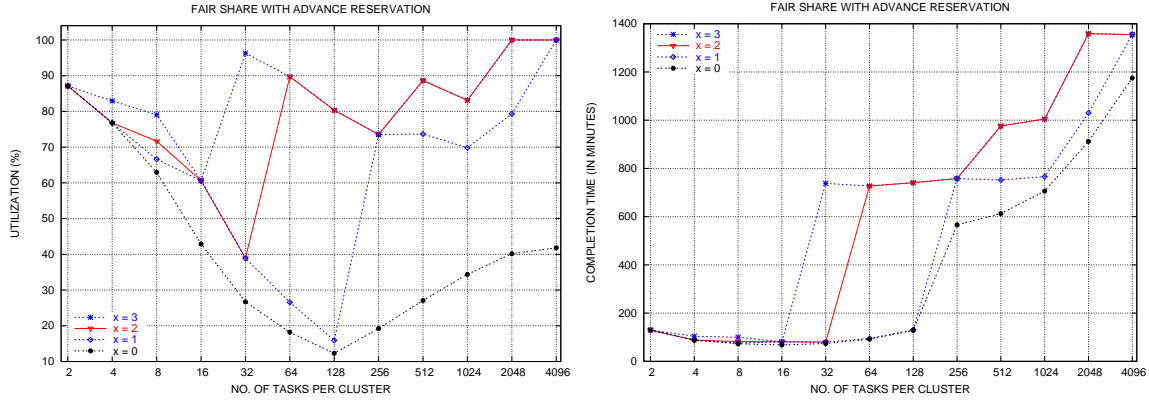


Figure 13: The resource utilization (left) and completion time (right) when requests are selected based on the value of the metric with $x = 0$ to $x = 3$ with Fair share and advance reservation policy.

Figure 13 shows similar results for the fair share scheduling policy. Another thing that is evident is that completion time reduction with provisioning does not always come at the expense of resource utilization. For example, the graphs with 2048 and 4096 tasks per cluster complete in 1359 and 1355 minutes with 100% resource utilization (Figure 13). The completion time of these graphs is 2312 and 2348 minutes without provisioning (Figure 9). Thus, for these graphs we obtain 41% and 42% reduction in completion time without sacrificing utilization.

7. Discussion

Resource provisioning can have a significant performance impact on the execution of workflows in a shared Grid environment. Based on the scheduling policies of the Grid site, the provisioning model used, and the workload it can lead to a significant reduction in the workflow completion time. The amount of reduction possible also depends on the scale and structure of the workflow. In general, large workflows like the original Montage workflow can obtain substantial reduction in the completion time even in the presence of backfilling policies. Several Grid sites use opportunistic schedulers that do not provide mechanisms for backfilling even for scheduling dedicated resources. At these sites, provisioning is useful even for small workflows.

It is true that most of the currently operational Grid sites do not provide online mechanisms for obtaining reservations or querying for the start times of the reservations. However, the simulation that we used to determine the earliest start time could very well have been done by the user with the information about the currently running and queued jobs obtained by querying the scheduler. Most schedulers provide such information. Dynamic provisioning only uses the estimates of the earliest start time and assumes no support for advance reservation at the remote site. Dynamic provisioning mechanisms such as Condor Glidein [3] can be used at sites where the worker machines are not behind a firewall. Current work is going on to address this limitation.

The utilization of the resources provisioned for execution a workflow can be suboptimal due to the structure of the workflow that do not allow all the resources to be used all the time. This suboptimal utilization results from different amount of parallelism at the different levels of the workflow. As we have shown, we can use a metric that incorporates utilization and hence allows us to tradeoff the utilization with the completion time. When there is a cost associated with the resources, increasing the resource

utilization allows us to execute the same workflow over a smaller number of resources leading to cost savings. We can also restructure the workflow so that the width of the workflow is as uniform as possible, for example, using the fixed number of clusters per level approach. As the experiments section show, restructuring based on fixed number of clusters per level in general had a better resource utilization than the fixed number of task per cluster.

When there are several possible sites are executing a workflow, the request selection mechanism for advance reservation can be used to select the site that guarantees the earliest completion time for the workflow. Hence it can be used as a mapping algorithm. Dynamic provisioning can be used to exploit this further by submitting requests to multiple sites and selecting the one that starts executing first, canceling the rest of the request. This type of multi site selection is not possible without provisioning. Another possibility is to use multiple requests at the same site to execute the workflow. For example, we selected the requested that gave the earliest completion time. However, in addition to this, we could have selected one more request with the earliest start time thus ensuring some progress in workflow execution till more resources become available. We plan to explore this in our future work.

The analysis done in this paper is deliberately simplified in order to focus our study. The calculated workflow execution time given a set of resource that we have used is theoretical. Also with no provisioning, we have integrated the workflow execution engine with the simulation controller leading to a fast execution for the workflow. In reality there are scheduling overheads and execution costs involved in submitted a job, monitoring it, determining its completion and analyzing the dependencies of the workflow that can make the real execution time an order of magnitude more than the simulated time particular for the fine granularity applications like Montage. Dynamic provisioning provides mechanisms for optimizing these costs. Thus in reality, even with backfilling, dynamic provisioning can lead to a significant reduction in workflow completion time for fine granularity workflows.

8. Survey

We did a survey on the support of advanced reservation capabilities at Grid sites (<http://pegasus.isi.edu/arsurvey.html>). 25 sites responded to the survey varying in the number of CPU's from 2 to 6874 (Teragrid [14]). Excluding the Teragrid [14] site, the rest of the sites had about 680 CPU's in average and thus had substantial computational resources. Out of these 25 sites, 12 of them provided no support for advanced reservations. Few of these sites indicated the use of opportunistic scheduling as the reason for not providing reservations. The rest of them appear to provide various degrees of support for advanced reservation. The majority of these sites require human intervention for setting up the reservation. Only two sites allow the user to establish a reservation without human intervention. The first one allow the use of command line tools for establishing the reservation (*setres* command in Maui) and the other uses online protocols [15]. The rest of the sites require sending an email or making a phone call to the system administrator. This was true even when the scheduler used at the site was capable of doing user initiated reservations. The lack of automated support for reservation seems to be because

- the lack of support for reservations in the tools being used.

- the site administrators do not perceive resource reservation as a frequent occurrence.
- the site administrators want to verify that the user has a real need for reservation that cannot be fulfilled by submitting jobs and discourage frivolous requests.
- the belief that reservations will reduce efficiency compared to batch scheduling approaches

The notice period required from the user varies from a few hours to a couple of weeks. The duration of the notice period depends on factors like the amount of resources requested, the nature of reservations (single site, multiple site), nature of resources (compute nodes, visualization systems), current and expected workload from non-reserving users and whether the user is local or remote. Most of the sites configure the resource management system in order to set up the reservation. This configuration is generally done by the system administrator after receiving the reservation request and verifying the authorization of the requester. The configuration change varies from explicitly setting up reservation (*setres* in Maui scheduler) to boosting shares for a particular user when fair sharing is being used. Some sites allow reservations only during the system scheduled downtime.

9. Related Work

The performance impact of advanced reservations on the metrics of interest to the system administrators e.g. resource utilization, mean queue time have been studied extensively [4, 5, 16, 17]. In this paper we examine the performance impact of reservation from the point of view of users who are interested in executing workflow class of application. Authors in [5] note that “*Current scheduling systems do not provide mechanisms to gain such simultaneous access without the help of human administrators of the computer systems*”. Even though the study was done in 2000 in the context of simultaneous access to multiple sites, it seems that the statement still holds true and human intervention is still required even for reserving resources on a single site (Section 8). The study done in [5] focuses on the affect of supporting reservation on the resource utilization and the average queue times as seen by the scheduler. It uses simulation to determine the time when the reservation can be granted, similar to what we have done. It also assumes a similar query interface to the scheduler for the user to determine the start time of the reservation.

The design and implementation of a scheduler that supports advanced reservations is described in [4]. It considers the problem of preventing users from taking start time advantage by using advanced reservations. A minimum notice period is imposed for advanced reservation that is a multiple of the average queue wait time. As we have observed in this paper, whether a advanced reservation presents a start time advantage depends not only on the scheduler policies and the current workload but also on the future workload and hence is difficult to decide in general. Thus we think that only when there is an expected gain by using advance reservation as we have shown for workflows, will the user take the trouble of setting up the reservation which may require having to go through a two phase commit protocol or something similar [4, 5]. A request scheduling algorithm that takes advantage of flexible resource reservation requests in order to improve the resource utilization is described in [17].

The impact of reservations for cross site reservations have been studied in [16]. The study describes a prototype metascheduler for scheduling *metajobs* that might require resources across sites. The metascheduler queries the local site scheduler for a range of available time slots where a reservation can be made. It shows that the system performance can be increased if support for online reservations can be provided instead of providing reservation during locked down periods. A common functionality in all the above mentioned studies has been the ability of the scheduler to determine the earliest start time of a request. Similar to us, [5] and [18] use simulation to determine the start time of a request. We use the requested time of the jobs in the simulation just like [5]. However, [18] uses the predicted runtime of jobs in the simulation in order to make more accurate estimates. They also describe a technique to make the simulation event driven that can greatly reduce the simulation time and make it feasible to do it in real time.

The request selection problem in order to minimize the completion time for moldable jobs have been studied earlier in parallel computing [6-8]. [7] introduces an application scheduler that chooses on the behalf of the user, which request to submit for a particular job. It also uses a simulation similar to ours in order to determine the best request. But the simulation is done by the user instead of the scheduler. Their work is mainly focused on the selection between the various possible requests which is only one part of our research. Our work is different from these works in several respects. Firstly we extend this work to workflows and cast a workflow as a moldable job. Secondly we consider the resource utilization of the resulting schedule as an explicit metric to be optimized. Lastly we also consider the performance of out-of-order scheduling policies while the previous work has mostly focused on FIFO like policies.

10. Conclusion and Future directions

In this paper, we have focused on the performance impact of using resource provisioning from a users perspective for workflow class of applications. Resource provisioning can be done using advance reservation or dynamic provisioning. The performance impact of advance reservations on the system metrics like resource utilization, and mean wait time have been the subject of much research in the past. We show that significant reduction in the completion time of the workflows is possible when the remote scheduler uses FIFO or Fair share policies. We show that dynamic provisioning that assumes no support for advance reservation at the remote sites can also be used with equal or more effectiveness than advance reservations. Finally we consider the issue of utilization of the provisioned resources and show the tradeoff between the resource utilization and the completion time.

The studies described in this paper were done with a particular workload trace gathered from the NCSA Teragrid cluster. The performance impact of resource provisioning depends on the workload, scheduling policies and the workflow characteristics. We believe the workload trace that we have used is representative of the typical workload at the supercomputing centers. In future, we would like to study the effect of using multiple provisioning requests for executing a workflow. In this paper, we have provisioned all the required resources up front. More generally the provisioning requests can be spread across time and sites and the decision has to be made on what to provision and when with respect to the costs associated with the resources, workflow completion time and the resulting resource utilization.

11. References

1. Buyya, R., ed. *Parallel Programming models and paradigms*. High Performance Cluster Computing: Architectures and Systems, ed. R.Buyya. Vol. 2. 1999, Prentice Hall PTR: NJ.
2. Deelman, E., et al., *Mapping Abstract Complex Workflows onto Grid Environments*. Journal of Grid Computing, 2003. **1**(1): p. 25-39.
3. Condor_Glidein, <http://www.cs.wisc.edu/condor/glidein>.
4. Cao, J. and F. Zimmermann. *Queue scheduling and advance reservations with COSY*. in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. 2004.
5. Smith, W., I. Foster, and V. Taylor. *Scheduling with advanced reservations*. in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. 2000.
6. Downey, A.B., *Using Queue Time Predictions for Processor Allocation in Proceedings of the Job Scheduling Strategies for Parallel Processing 1997* Springer-Verlag. p. 35-57
7. Cirne, W. and F. Berman, *Using Moldability to Improve the Performance of Supercomputer Jobs*. Journal of Parallel and Distributed Computing, 2002. **62**(10): p. 1571-1601.
8. Feitelson, D.G., et al., *Theory and Practice in Parallel Job Scheduling in Proceedings of the Job Scheduling Strategies for Parallel Processing 1997* Springer-Verlag. p. 1-34
9. Jackson, D.B., H.L. Jackson, and Q.O. Snell. *Simulation based HPC workload analysis*. in *Parallel and Distributed Processing Symposium., Proceedings 15th International*. 2001.
10. Hwang, J.-J., et al., *Scheduling precedence graphs in systems with interprocessor communication times* SIAM J. Comput. , 1989 **18** (2): p. 244-257
11. Berriman, G.B., et al. *Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand*. in *SPIE Conference 5487: Astronomical Telescopes*. 2004.
12. Eager, D.L., J. Zahorjan, and E.D. Lazowska, *Speedup versus efficiency in parallel systems*. Computers, IEEE Transactions on, 1989. **38**(3): p. 408-423.
13. Kumar Jain, K. and V. Rajaraman, *Lower and upper bounds on time for multiprocessor optimal schedules*. Parallel and Distributed Systems, IEEE Transactions on, 1994. **5**(8): p. 879-886.
14. Catlett, C. *The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility*. in *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*. 2002.
15. Mumtaz Siddiqui, T.F. *GridARM: Askalon's Grid Resource Management System*. in *European Grid Conference*. 2005. Amsterdam: Springer Verlag.
16. Snell, Q., et al., *The Performance Impact of Advance Reservation Meta-scheduling in Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing 2000* Springer-Verlag. p. 137-153

17. Chiu, P.L., Y. Chen, and K.H. Lee. *A request scheduling algorithm to support flexible resource reservations in advance.* in *Electrical and Computer Engineering, 2004. Canadian Conference on.* 2004.
18. Li, H., et al. *Predicting job start times on clusters.* in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on.* 2004.