Chapter 1

IMPROVING RESOURCE SELECTION AND SCHEDULING USING PREDICTIONS

Warren Smith
Computer Sciences Corporation
NASA Ames Research Center
wwsmith@nas.nasa.gov

Abstract

The introduction of computational grids has resulted in several new problems in the area of scheduling that can be addressed using predictions. The first problem is selecting where to run an application on the many resources available in a grid. Our approach to help address this problem is to provide predictions of when an application would start to execute if submitted to specific scheduled computer systems. The second problem is gaining simultaneous access to multiple computer systems so that distributed applications can be executed. We help address this problem by investigating how to support advance reservations in local scheduling systems. Our approaches to both of these problems are based on predictions for the execution time of applications on space-shared parallel computers. As a side effect of this work, we also discuss how predictions of application run times can be used to improve scheduling performance.

Keywords: Run time prediction, wait time prediction, performance prediction, scheduling, grid computing.

1. Introduction

The existence of computational grids allows users to execute their applications on a variety of different computer systems. An obvious problem that arises is how to select a computer system to run an application. Many factors go into making this decision: The computer systems that the user has access to, the user's remaining allocations on

these systems, the cost of using different systems, the location of data sets for the experiment, how long the application will execute on different computers, when the application will start executing, and so on. We wish to help users make this decision, so in Section 1.3, we discuss approaches to predicting when a scheduling system for a space shared parallel computer will start executing an application.

The large number of resources available in computational grids leads users to want to execute applications that are distributed across multiple resources; such as running a large simulation on 2 or more supercomputers. The difficulty with this is that different resources may have different scheduling systems without any mechanisms to guarantee that an application obtains simultaneous access to the different resources. To address this problem, Section 1.5 describes ways to incorporate advance reservations into scheduling systems and analyze their performance. Users can use advance reservations to ask for resources from multiple schedulers at the same time in the future and obtain simultaneous access. As a side effect of this work, in Section 1.4 we discuss how to improve the performance of scheduling systems even when no reservations are needed.

We base much of the work above on techniques to predict the execution time of applications on space shared parallel computers. We begin this chapter by describing two techniques to calculate predictions of application run times and discuss their performance.

2. Run Time Predictions

There have been many efforts to predict the execution time of serial and parallel applications. We can roughly classify prediction techniques by whether the resources that are used are shared or dedicated, the type and detail of application and resource models that are used, and the type of prediction technique used.

There have been many efforts to predict the execution time of applications on shared computer systems [4, 5, 25, 19] and dedicated computer systems [21, 22, 6, 12, 14, 15]. Almost all of the techniques referenced above use very simple models of an application. Typically either the information provided to the scheduler or this information plus a few application-specific parameters [15]. An exception is the work by Schopf [19, 18] where high-level structural models of applications were created with the assistance of the creators of the applications.

The type of prediction technique that is used can generally be separated into statistical, which uses statistical analysis of applications that have completed, and analytical, which constructs equations describing

application execution time. Statistical approaches use time series analysis [25, 5], categorization [21, 6, 12], and instance-based learning or locally weighted learning [15, 14, 22]. Analytical approaches develop models by hand [19] or use automatic code analysis or instrumentation [23].

We present and analyze the performance of two techniques that we have developed. Both techniques are statistical and only use the information provided when an application is submitted to a scheduler to make predictions. The first technique uses categories to form predictions and the second technique uses instance-based learning.

2.1 Categorization Prediction Technique

Our first approach to predicting application run times is to derive run time predictions from historical information of previous similar runs. This approach is based on the observation [21, 6, 9, 12] that similar applications are more likely to have similar run times than applications that have nothing in common.

One difficulty in developing prediction techniques based on similarity is that two applications can be compared in many ways. For example, we can compare applications on the basis of application name, submitting user name, executable arguments, submission time, and number of nodes requested. When predicting application run times in this work, we restrict ourselves to those values recorded in traces obtained from various supercomputer centers. The workload traces that we consider originate from 3 months of data from an IBM SP at Argonne National Laboratory (ANL), 11 months of data from an IBM SP at the Cornell Theory Center (CTC), and 2 years of data from an Intel Paragon at the San Diego Supercomputer Center (SDSC). The characteristics of jobs in each workload vary but consist of a relatively small set of characteristics such as user name application name, queue name, run time, and so on.

Our general approach to defining similarity is to use job characteristics to define templates that identify a set of categories to which applications can be assigned. For example, the template (queue,user) specifies that applications are to be partitioned by queue and user. On the SDSC Paragon, this template generates categories such as (q16m, wsmith), (q64l, wsmith), and (q16m, foster).

We find that categorizing discrete characteristics (such as user name) in the manner just described works reasonably well. On the other hand, the number of nodes is an essentially continuous parameter, and so we prefer to introduce an additional parameter into our templates, namely, a node range size that defines what ranges of requested number of nodes

are used to decide whether applications are similar. For example, the template (u, n=4) specifies a node range size of 4 and generates categories (wsmith, 1-4 nodes) and (wsmith, 5-8 nodes).

In addition to the characteristics of jobs contained in the workloads, a *maximum history*, *type of data* to store, and *prediction type* are also defined for each run time prediction template. The maximum history indicates the maximum number of data points to store in each category generated from a template. The type of data is either an actual run time or a relative run time. A relative run time incorporates information about user-supplied run time estimates by storing the ratio of the actual run time to the user-supplied estimate (the amount of time the nodes are requested for). The prediction type determines how a run time prediction is made from the data in each category generated from a template. We considered four prediction types in our previous work: a mean, a linear regression, an inverse regression, and a logarithmic regression [7]. We found that the mean is the single best predictor [20], so this work uses only means to form predictions. We also take into account running time, how long an application has been running when a prediction is made, when making predictions by ignoring data points from a category that have a run time less than this running time.

Once a set of templates has been defined (using a search process described later), we simulate a workload of application predictions and insertions. For each insertion, an application is added to the categories that contain similar applications. For each prediction, an execution time and a confidence interval is calculated. A prediction is formed from each similar category of applications and the prediction with the smallest confidence interval is selected to be the prediction for the application.

We use a genetic algorithm search to identify good templates for a particular workload. While the number of characteristics included in our searches is relatively small, the fact that effective template sets may contain many templates means that an exhaustive search is impractical. Genetic algorithms are a probabilistic technique for exploring large search spaces, in which the concept of cross-over from biology is used to improve efficiency relative to purely random search [13]. A genetic algorithm evolves individuals over a series of generations. Our individuals represent template sets. Each template set consists of between 1 and 10 templates, and we encode the previously described information in each template. The process for each generation consists of evaluating the fitness of each individual in the population, selecting which individuals will be mated to produce the next generation, mating the individuals, and mutating the resulting individuals to produce the next generation. The process then repeats until a stopping condition is met. The stopping

Table 1.1. Performance of our categorization technique versus those of Gibbons and Downey.

Workload	Mean Error (minutes)			Mean Run
	Ours	Gibbons	Downey	Time (minutes)
ANL	34.52	75.26	97.01	97.08
CTC	98.28	124.06	179.46	182.49
SDSC95	43.20	74.05	82.44	108.16
SDSC96	47.47	122.55	102.04	166.85

condition we use is that a fixed number of generations are processed. Further details of our searches are available in [20]

The accuracy of the prediction parameters found by our searches are shown in the second column of Table 1.1. The prediction error is 39 percent on average and ranges from 28 percent for the SDSC96 workload to 54 percent for the CTC workload. We compare these results to the run time prediction performance technique proposed by Gibbons [12] and the run time prediction technique used by Downey [6] to predict how long jobs will wait at the head of a scheduling queue before beginning to execute. Both of these techniques categorize applications that have completed executing, find categories that are similar to an application whose run time is to be predicted, and then form a prediction from these categories. The categories and techniques used to calculate predictions differ between the two techniques and differ from our technique. Table 1.1 shows that our predictions have between 21 and 61 percent lower mean error than the Gibbons' approach and 45 to 64 percent lower mean error than the better of Downey's two techniques.

2.2 Instance-Based Learning Approach

In our second approach, we predict the execution time of applications using instance-based learning (IBL) techniques that are also called locally weighted learning techniques [1, 17]. In this type of technique, a database of experiences, called an experience base, is maintained and used to make predictions. Each experience consists of input and output features. Input features describe the conditions under which an experience was observed and the output features describe what happened under those conditions. Each feature consists of a name and a value where the value is of a simple type such as integer, floating-point number, or string. In this work, the input features are the same ones as used in our first approach: The user who submitted the job, the application

that was executed, the number of CPUs requested, and so on. The execution time of the job is the only output feature of the experience.

When a prediction is to be performed, a query point consisting of input features is presented to the experience base. The data points in the experience base are examined to determine how relevant they are to the query where relevance is determined using the distance between an experience and the query. There are a variety of distance functions that can be used [24] and we choose to use the Heterogeneous Euclidean Overlap Metric [24]. This distance function can be used on features that are linear (numbers) or nominal (strings). We require support for nominal values because important features such as the names of executables, users, and queues are nominal. As a further refinement, we perform feature scaling to stretch the experience space and increase the importance that certain features are similar.

Once we know the distance between experiences and a query point, the next question to be addressed is how we calculate estimates for the output features of the query point. For linear output features, such as execution time, our approach is to use a distance-weighted average of the output features of the experiences to form an estimate. We choose to use a Gaussian function to form this distance-weighted average. Further, we include a factor, called the kernel width, so that we can compact or stretch the Gaussian function to give lower or higher weights to experiences that are farther away.

To perform an estimate, we must select values for the parameters discussed above along with the maximum experience base size and the number of nearest neighbors (experiences) to use. Our approach to determine the best values for these parameters is to once again perform a genetic algorithm search [13] to select values that minimize the prediction error.

Table 1.2 shows a comparison of our categorization technique and our instance-based learning technique using the same trace data that we used to evaluate our first technique. The table shows that at the current time, our instance-based learning technique has an error which is 44 percent of mean run times and 59 percent lower than the user estimates available in the ANL and CTC workloads. The user estimates for the ANL and CTC workloads are provided along with each job. Our IBL technique has an 89 percent lower error than the run time estimates that can be derived from the SDSC workloads. This system has many queues with different resource usage limits. We derive the run time limits for each queue by examining the workloads and finding the longest running job submitted to each queue.

Table 1.2. A comparison of our two execution time prediction techniques.

Workload	IBL Mean Error (minutes)	Categorization Mean Error (minutes)	Mean Error of User Estimate (minutes)	Mean Run Time (minutes)
ANL	36.93	34.52	104.35	97.08
CTC	103.75	98.28	222.71	182.49
SDSC95	51.61	43.20	466.49	108.16
SDSC96	52.79	47.47	494.25	166.85

Table 1.2 also shows that the error of our categorization technique is currently 10 percent lower than our IBL technique. There are several possible reasons for this result. First, we performed more extensive searches to find the best parameters used in the categorization technique. Second, the categorization technique essentially uses multiple distance functions and selects the best results obtained after using each of these functions instead of the single distance function used by our IBL approach. In future work, we will evaluate how well our IBL approach performs when using multiple distance functions.

3. Queue Wait Time Predictions

On many high-performance computers, a request to execute an application is not serviced immediately but is placed in a queue and serviced only when the necessary resources are released by running applications. On such systems, predictions of how long queued requests will wait before being serviced are useful for a variety of tasks. For example, predictions of queue wait times can guide a user in selecting an appropriate queue or, in a computational grid, to an appropriate computer [11]. Wait time predictions are also useful in a grid computing environment when trying to submit multiple requests so that the requests all receive resources simultaneously [2]. A third use of wait time predictions is to plan other activities in conventional supercomputing environments.

We examine two different techniques for predicting how long applications wait until they receive resources in this environment. Our first technique for predicting wait times in scheduling systems is to predict the execution time for each application in the system (using the categorization technique presented in Section 1.2) and then to use those predicted execution times to drive a simulation of the scheduling algorithm. This allows us to determine the start time of every job in the scheduling system. The advantage of this technique is that, for certain scheduling algorithms and accurate run time predictions, it can poten-

tially provide very accurate wait time predictions. A disadvantage is that if the scheduling algorithm is such that the start times of applications in the queues depend on applications that have not yet been submitted to the queues, the wait time predictions will not be very accurate. A second disadvantage of this technique is that it requires detailed knowledge of the scheduling algorithm used by the scheduling system.

Our second wait time prediction technique predicts the wait time of an application by using the wait times of applications in the past that were in a similar scheduler state. For example, if an application is in a queue with four applications ahead of it and three behind it, how long did applications in this same state wait in the past? This approach uses the same mechanisms as our approach to predicting application execution times with different characteristics used to describe the conditions we are predicting.

3.1 Scheduling Algorithms

We use three basic scheduling algorithms in this work: first-come first-served (FCFS), least work first (LWF), and conservative backfill [3, 10] with FCFS queue ordering. In the FCFS algorithm, applications are given resources in the order in which they arrive. The application at the head of the queue runs whenever enough nodes become free. The LWF algorithm also tries to execute applications in order, but the applications are ordered in increasing order using estimates of the amount of work (number of nodes multiplied by estimated wall clock execution time) the application will perform.

The conservative backfill algorithm is a variant of the FCFS algorithm. The difference is that the backfill algorithm allows an application to run before it would in FCFS order if it will not delay the execution of applications ahead of it in the queue (those that arrived before it). When the backfill algorithm tries to schedule applications, it examines every application in the queue, in order of arrival time. If an application can run (there are enough free nodes and running the application will not delay the starting times of applications ahead of it in the queue), it is started. If an application cannot run, nodes are reserved for it at the earliest possible time. This reservation is only a placeholder to make sure that applications behind it in the queue do not delay it; the application may actually start before this time.

3.2 Predicting Queue Wait Times: Technique 1

Our first wait time prediction technique simulates the actions performed by a scheduling system using predictions of the execution times

Table 1.3. Wait time prediction performance using actual and maximum run times.

Workload	Scheduling Algorithm	Wait Time Prediction Error		Mean Wait Time (minutes)
		Actual Run Times (minutes)	Maximum Run Times (minutes)	
ANL	FCFS	0.00	996.67	535.84
ANL	LWF	37.14	97.12	86.71
ANL	Backfill	5.84	429.05	177.29
CTC	FCFS	0.00	125.36	97.94
CTC	LWF	4.05	9.86	10.49
CTC	Backfill	2.62	51.16	26.93
SDSC95	FCFS	0.00	162.72	55.16
SDSC95	LWF	5.83	28.56	14.95
SDSC95	Backfill	1.12	93.81	28.17
SDSC96	FCFS	0.00	47.83	16.61
SDSC96	LWF	3.32	14.19	7.88
SDSC96	Backfill	0.30	39.66	11.33

of the running and waiting applications. We simulate the FCFS, LWF, and backfill scheduling algorithms and predict the wait time of each application when it is submitted to the scheduler.

Table 1.3 shows the wait time prediction performance when actual or maximum run times are used during prediction. The actual run times are the exact running times of the applications, which are not known ahead of time in practice. The maximum run time is the amount of time a job requests the nodes for and is when an application should be terminated if it hasn't already completed. This data provides upper and lower bounds on wait time prediction accuracy and can be used to evaluate our prediction approach. When using actual run times, there is no error for the FCFS algorithm because later arriving jobs do not affect the start times of the jobs that are currently in the queue. For the LWF and backfill scheduling algorithms, wait time prediction error does occur because jobs that have not been enqueued can affect when the jobs currently in the queue can run. This effect is larger for the LWF results where later-arriving jobs that wish to perform smaller amounts of work move to the head of the queue. When predicting wait times using actual run times, the wait time prediction error for the LWF algorithm is between is between 34 and 43 percent. There is a very high built-in error when predicting queue wait times of the LWF algorithm with this technique because there is a higher probability that applications that have not yet been submitted will affect when already submitted appli-

Table 1.4. Wait time prediction performance of our two techniques.

Workload	Scheduling Algorithm	Technique 1 Run Time Prediction Error (minutes)	Technique 1 Wait Time Prediction Error (minutes)	Technique 2 Wait Time Prediction Error (minutes)
ANL	FCFS	38.26	161.49	260.36
ANL	LWF	54.11	44.75	76.78
ANL	Backfill	46.16	75.55	130.35
CTC	FCFS	125.69	30.84	76.18
CTC	LWF	145.28	5.74	9.80
CTC	Backfill	145.54	11.37	22.95
SDSC95	FCFS	53.14	20.34	39.79
SDSC95	LWF	58.98	8.72	13.67
SDSC95	Backfill	57.87	12.49	25.50
SDSC96	FCFS	55.92	9.74	10.55
SDSC96	LWF	54.27	4.66	6.83
SDSC96	Backfill	54.82	5.03	9.26

cations will start. There is also a small error (3 - 4%) when predicting the wait times for the backfill algorithm.

The table also shows that the wait time prediction error of the LWF algorithm when using actual run times as run time predictors is 59 to 80 percent better than the wait time prediction error when using maximum run times as the run time predictor. For the backfill algorithm, using maximum run times results in between 96 and 99 percent worse performance than using actual run times. Maximum run times are provided in the ANL and CTC workload and are implied in the SDSC workloads because each of the queues in the two SDSC workloads have maximum limits on resource usage.

The third column of Table 1.4 shows that our run time prediction technique results in run time prediction errors that are from 33 to 86 percent of mean application run times and the fourth column shows that the wait time prediction errors that are from 30 to 59 percent of mean wait times. The results also show that using our run time predictor result in mean wait time prediction errors that are 58 percent worse than when using actual run times for the backfill and LWF algorithms but 74 percent better than when using maximum run times.

3.3 Predicting Queue Wait Times: Technique 2

Our second wait time prediction technique uses historical information on scheduler state to predict how long applications will wait until they receive resources. This is an instance of the same categorization prediction approach that we use to predict application run times in Section 1.2. We selected scheduler state characteristics that describe the parallel computer being scheduled, the application whose wait time is being predicted, the time the prediction is being made, the applications that are waiting in the queue ahead of the application being predicted, and the applications that are running to use when making predictions. The characteristics of scheduler state are continuous parameters, similar to the number of nodes specified by an application. Therefore, a range size is used for all of these characteristics.

The fifth column of Table 1.4 shows the performance of this wait time prediction technique. The data shows that the wait time prediction error is 42 percent worse on average than our first wait time prediction technique. One trend to notice is that the predictions for the FCFS scheduling algorithm are the most accurate for all of the workloads, the predictions for the backfill algorithm are all the second most accurate, and the predictions for the LWF algorithm are the least accurate. This is the same pattern when the first wait time prediction technique is used with actual run times. This indicates that our second technique is also affected by not knowing what applications will be submitted in the near future.

4. Scheduling Using Run Time Predictions

Many scheduling algorithms use predictions of application execution times when making scheduling decisions [3, 10, 8]. Our goal in this section is to improve the performance of the LWF and backfill scheduling algorithms that use run time predictions to make scheduling decisions. We measure the performance of a scheduling algorithm using *utilization*, the average percent of the machine that is used by applications, and *mean wait time*, the average amount of time that applications wait before receiving resources.

Table 1.5 shows the performance of the scheduling algorithms when the actual run times are used as run time predictors and when maximum run times are used as run time predictors. These numbers give us upper and lower bounds on the scheduling performance we can expect. The data shows that while maximum run times are not very accurate predictors, this has very little effect on the utilization of the simulated

Table 1.5. Scheduling performance using actual and maximum run times.

Workload	Scheduling Algorithm	Actual Run Times		Maximum Run Times	
		Utilization (percent)	Mean Wait Time (minutes)	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	70.34	61.20	70.70	83.81
ANL	Backfill	71.04	142.45	71.04	177.14
CTC	LWF	55.18	11.15	55.18	36.95
CTC	Backfill	55.18	27.11	55.18	123.91
SDSC95	LWF	41.14	14.48	41.14	14.95
SDSC95	Backfill	41.14	21.98	41.14	28.20
SDSC96	LWF	46.79	6.80	46.79	7.88
SDSC96	Backfill	46.79	10.42	46.79	11.34

parallel computers. Predicting run times with actual run times when scheduling results in an average of 30 percent lower mean wait times.

To evaluate how well our run time predictions can improve scheduling performance, the first thing we need to determine is what template sets to use to predict application run times. We initially tried to minimize the run time prediction error for workloads generated by running the scheduling algorithms using maximum run times as predictors and recording all predictions that were made. We were not satisfied with the scheduling performance obtained using the parameters obtained by searching over these workloads. So, instead of attempting to minimize run time prediction error, we perform scheduling simulations using different run time prediction parameters and attempt to directly minimize wait times. We do not attempt to maximize utilization because utilization only changes very slightly when different template sets are used or even when a different scheduling algorithm is used.

Table 1.6 shows the performance of the results of these searches. As expected, our run time prediction has minimal impact on utilization. Using our run time predictions does decrease mean wait time by an average of 25 percent over using maximum wait times. These mean wait times are 5 percent more than the wait times achieved when using actual run times.

5. Scheduling With Advance Reservations

Some applications have very large resource requirements and would like to use resources from multiple parallel computers to execute. In this section, we describe one solution to this *co-allocation* problem: Advance

Table 1.6. Scheduling performance using our run time prediction technique.

Workload	Scheduling Algorithm	Mean Run Time Prediction Error (minutes)	Scheduling Utilization (percent)	Scheduling Mean Wait Time (minutes)
ANL	LWF	72.88	70.75	69.31
ANL	Backfill	116.52	71.04	174.14
CTC	LWF	182.96	55.18	10.58
CTC	Backfill	61.73	55.18	28.39
SDSC95	LWF	142.01	41.14	14.82
SDSC95	Backfill	38.37	41.14	22.21
SDSC96	LWF	112.13	46.79	7.43
SDSC96	Backfill	47.06	46.79	10.56

reservation of resources. Reservations allow a user to request resources from multiple scheduling systems at a specific time and thus gain simultaneous access to sufficient resources for their application.

We investigate several different ways to add support for reservations into scheduling systems and evaluate their performance. We evaluate scheduling performance using utilization and mean wait time, as in the previous section, and also *mean offset from requested reservation time*; the average difference between when the users initially want to reserve resources for each application and when they actually obtain reservations. The mean offset from requested reservation time measures how well the scheduler performs at satisfying reservation requests.

In this section, we use these metrics to evaluate a variety of techniques for combining scheduling from queues with reservation. There are several assumptions and choices to be made when doing this. The first is whether applications are restartable. Most scheduling systems currently assume that applications are not restartable (a notable exception is the Condor system described in [16] and Chapter ??). We evaluate scheduling techniques when applications both can and cannot be restarted. We assume that when an application is terminated, intermediate results are not saved and applications must restart execution from the beginning. We also assume that a running application that was reserved cannot be terminated to start another application. Further, we assume that once the scheduler agrees to a reservation time, the application will start at that time.

If we assume that applications are not restartable and the scheduler must fulfill it once it is made, then we must use maximum run times when predicting application execution times to ensure that nodes are available.

The resulting scheduling algorithms essentially perform backfilling. This approach is discussed in Section 1.5.2

If applications are restartable, there are more options for the scheduling algorithm and this allows us to improve the scheduling performance. First, the scheduler can use run time predictions other than maximum run times. Second, there are many different ways to select which running applications from the queue to terminate to start a reserved application. Details of these options and their performance are presented in Section 1.5.3

Due to space limitations, we only summarize the performance data we collected for the techniques presented in this section. Please see [20] for a more comprehensive presentation of performance data.

5.1 Reservation Model

In our model, a reservation request consists of the number of nodes desired, the maximum amount of time the nodes will be used, the desired start time, and the application to run on those resources. We assume that the following procedure occurs when a user wishes to submit a reservation request:

1 The user requests that an application run at time T_r on N nodes for at most M amount of time.

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

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

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

5.2 Nonrestartable Applications

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

cute longer than expected and prevent other reserved applications from starting. This means that only maximum run times can be used when making scheduling decisions.

A scheduler decides when an application from a queue can be started using an approach similar to the backfill algorithm: The scheduler creates a timeline of when it believes the nodes of the system will be used in the future. First, the scheduler adds the currently running applications and the reserved applications to the timeline using their maximum run times. Then, the scheduler attempts to start applications from the queue using the timeline and the number of nodes and maximum run time requested by the application to make sure that there are no conflicts for node use. If backfilling is not being performed, the timeline is still used when starting an application from the head of the queue to make sure that the application does not use any nodes that will be needed by reservations.

To make a reservation, the scheduler first performs a scheduling simulation of applications currently in the system and produces a timeline of when nodes will be used in the future. This timeline is then used to determine when a reservation for an application can be made.

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

5.2.1 Effect of Reservations on Scheduling. We begin by evaluating the impact on the mean wait times of queued applications when reservations are added to our workloads. We assume the best case for queued applications: When reservations arrive, they cannot be scheduled so that they delay any currently queued applications.

We add reservations to our existing workloads by randomly converting either ten or twenty percent of the applications to be reservations with requested reservation times randomly selected between zero and two hours in the future. We find that adding reservations increases the wait times of queued applications in almost all cases. For all of the workloads, queue wait times increase an average of 13 percent when 10 percent of the applications are reservations and 62 percent when 20 percent of the applications are reservations. Our data also shows that if we perform

backfilling, the mean wait times increase by only 9 percent when 10 percent of the applications are reservations and 37 percent when 20 percent of the applications are reservations. This is a little over half of the increase in mean wait time when backfilling is not used. Further, there is a slightly larger increase in queue wait times for the LWF queue ordering than for the FCFS queue ordering.

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

The performance is what is expected in general: The offset is larger when there are more reservations. For 10 percent reservations, the mean difference from requested reservation time is 211 minutes. For 20 percent reservations, the mean difference is 278 minutes.

Our data also shows that the difference between requested reservation times and actual reservation times is 49 percent larger when FCFS queue ordering is used. The reason for this may be that LWF queue ordering will execute the applications currently in the queue faster than FCFS. Therefore, reservations can be scheduled at earlier times.

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

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

As expected, if more queued applications can be delayed when a reservation request arrives, then the wait times are generally longer and the offsets are smaller. On average, for the ANL workload, decreasing the

percent of queued applications with priority from 100 to 50 percent increases mean wait time by 7 percent and decreases mean offset from requested reservation times by 39 percent. Decreasing the percent of queued application with priority from 100 to 0 percent increases mean wait time by 22 percent and decreases mean offset by 89 percent. These results for the change in the offset from requested reservation time are representative of the results from the other three workloads: as fewer queued applications have priority, the reservations are closer to their requested reservations.

5.3 Restartable Applications

If we assume that applications can be terminated and restarted, then we can improve scheduling performace by using run time predictions other than maximum run times when making scheduling decisions. We use our categorization run time prediction technique that we described in Section 1.2.

The main choice to be made in this approach is how to select which running applications to terminate when CPUs are needed to satisfy a reservation. There are many possible ways to select which running applications that came from a queue should be terminated to allow a reservation to be satisfied. We choose a rather simple technique where the scheduler orders running applications from queues in a list based on termination cost and moves down the list stopping applications until enough CPUs are available. Termination cost is calculated using how much work (number of CPUs multiplied by wall clock run time) the application has performed and how much work the application is predicted to still perform. Our data shows that while the appropriate weights for work performed and work to do vary from workload to workload, in general, the amount of work performed is the more important factor.

We performed scheduling simulations with restartable applications using the ANL workload. When we compare this performance to the scheduling performance when applications are not restartable, we find that the mean wait time decreases by 7 percent and the mean difference from requested reservation time decreases by 55 percent. There is no significant effect on utilization. This shows that there is a performance benefit if we assume that applications are restartable, particularly in the mean difference from requested reservation time.

6. Summary

The availability of computational grids and the variety of resources available through computational grids introduce two problems that we

seek to address through predictions. The first problem is selecting where to run an application in a grid. The second problem is gaining simultaneous access to multiple resources so that a distributed application can be executed.

Our approaches to both of these problems are based on predictions of the execution time of applications. We propose and evaluate two techniques for predicting these execution times. Our first technique categorizes applications that have executed in the past and forms a prediction for an application using categories of similar applications. Our second technique uses historical information and an instance-based learning approach. We find that our categorization approach is currently the most accurate and is more accurate than the estimates provided by users or the techniques presented by two other researchers.

We address the problem of selecting where to run an application in a computational grid by proposing two approaches to predicting scheduling queue wait times. The first technique uses run time predictions and performs scheduling simulations. The second technique characterizes the state of a scheduler and the application whose wait time is being predicted and uses historical information of wait times in these similar states to produce a wait time prediction. We find that our first technique has a lower prediction error of between 30 and 59 percent of the mean wait times.

We address the problem of gaining simultaneous access to distributed resources by describing several ways to modify local scheduling systems to provide advance reservations. We find that if we cannot restart applications, we are forced to use maximum run times as predictions when scheduling. If we can restart applications, then we can use our run time predictions when scheduling. We find that supporting advance reservations does increase the mean wait times of applications in the scheduling queues but this increase is smaller if we are able to restart applications. As a side effect of this work, we find that even when there are no reservations, we can improve the performance of scheduling algorithms by using more accurate run time predictions.

Acknowledgments

We wish to thank Ian Foster and Valerie Taylor who investigated many of the problems discussed here with the author. This work has been supported by Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research of the U.S. Department of Energy, the NASA Information

Technology program and the NASA Computing, Information and Communications Technology program.

References

[1] Christopher Atkeson, Andrew Moore, and Stefan Schaal. *Locally Weighted Learning.* Artificial Intelligence Review, 11, 1997., 1997.

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

[3] D.A.Lifka. *The ANL/IBM SP Scheduling System.* In D.G.Feitelson and L.Rudolph,editors,IPPS '95 Workshop:Job Scheduling Strategies for Parallel Processing ,pages 295 -303.Springer -Verlag,Lecture Notes in Computer Science LNCS 949,1995., 1995.

[4] Murthy Devarakonda and Ravishankar Iyer. *Predictability of Process Resource Usage: A Measurement-Based Study on UNIX.* IEEE Transactions on Software Engineering, 15 (12 :1579-1586, December 1989., 1989.

[5] Peter Dinda. *Online Prediction of the Running Time of Tasks.* In Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, 2001., 2001.

[6] Allen Downey. *Predicting (queue Times on Space-Sharing Parallel Computers.* In International Parallel Processing Symposium, 1997., 1997.

[7] N. R. Draper and H. Smit. *Applied Regression Analysis, 2nd Edition.* John Wiley and Sons, 1981., 1981.

[8] Dror Feitelson. *A Survey of Scheduling in Multiprogrammed Parallel Systems.* Technical Report RC 19790, IBM T.J. Watson Research Center, October 1995., 1995.

[9] Dror Feitelson and Bill Nitzberg. *Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860.* Lecture Notes on Computer Science, 949, 1995., 1995.

[10] Dror Feitelson and Ahuva Weil. *Utilization and Predictability in Scheduling the IBM SP2 with Backfilling.* In Procedings of 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing, 1998., 1998.

[11] I. Foster, C. Kesselman, and eds. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1998., 1998.

[12] Richard Gibbons. *A Historical Application Profiler for Use by Parallel Scheculers.* Lecture Notes on Computer Science, 1297, 1997., 1997.

[13] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989., 1989.

[14] M. Iverson, F. Ozguner, and L. Potter. *Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment.* In Proceedings of the IPPS/SPDP'99 Heterogeneous Computing Workshop, 1999., 1999.

[15] N. Kapadia, J. Fortes, and C. Brodley. *Predictive Application Performance Modeling in a Computational Grid Environment.* In Proceedings of the 8th High Performance Distributed Computing Conference, 1999., 1999.

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

[17] J. Schneider and A. Moore. *A Locally Weighted Learning Tutorial using Vizier 1.0.* Technical Report CMU-RI-TR-00-18, Robitics Institute, Carnegie Mellon University, Febuary 2000., 2000.

[18] Jennifer Schopf. *Structural Prediction Models for High Performance Distributed Applications.* In Proceedings of the 1997 Cluster Computing Conference, 1997., 1997.

[19] Jennifer Schopf and Francine Berman. *Performance Prediction in Production Environments.* In l4th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing, 1998., 1998.

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

[21] Warren Smith, Ian Foster, and Valerie Taylor. *Predicting Application Run Times Using Historical Information.* Lecture Notes on Computer Science, 1459, 1998., 1998.

[22] Warren Smith and Parkson Wong. *Resource Selection Using Execution and Queue Wait Time Predictions.* Technical Report NAS02-003, NASA Ames Research Center, July 2002., 2002.

[23] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan, Mark Hereld, Ivan Judson, and Rick Stevens. *Prophesy: Automating the Modeling Process.* Third International Workshop on Active Middleware Services, 2001.

[24] D. R. Wilson and T. R. Martinez. *Improved Heterogeneous Distance Functions.* Journal of Artificial Intelligence Research, 6, 1997., 1997.

[25] Rich Wolski, Neil Spring, and Jim Hayes. *Predicting the CPU Availability of Time-Shared Unix Systems.* In Proceedings of the 8th

IEEE International Symposium on High Performance Distributed Computing, 1999., 1999.