Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance

Warren Smith*† Valerie Taylor† Ian Foster*

{wsmith, foster}@mcs.anl.gov
taylor@ece.nwu.edu

*Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
http://www.mcs.anl.gov

†Electrical and Computer Engineering Department
Northwestern University
Evanston, IL 60208
http://www.ece.nwu.edu

Abstract

On many computers, a request to run a job is not serviced immediately but instead is placed in a queue and serviced only when resources are released by preceding jobs. In this paper, we build on run-time prediction techniques that we developed in previous research to explore two problems. The first problem is to predict how long applications will wait in a queue until they receive resources. We show that run-time estimates can be used for this and that using our run-time estimates result in more accurate wait-time predictions than when the run-time prediction techniques of other researches are used. The second problem we investigate is improving scheduling performance. We use run-time predictions to improve the performance of the least work first and backfill scheduling algorithms. We find that using our run-time predictor results in lower mean wait times for the workloads with higher offered loads when compared to alternative run-time predictors.

1 Introduction

On many high-performance computers, a request to execute an application is not serviced immediately but instead is placed in a queue and serviced only when resources are released by running applications. We examine two separate problems in this environment. First, we predict how long applications will wait until they execute. These estimates of queue wait times are useful to guide resource selection when several systems are available [7], to co-allocate resources from multiple systems [2], to schedule other activities, etc. Our technique for predicting queue wait times is to use predictions of application execution times along with the scheduling algorithms to simulate the actions made by a scheduler and determine when applications will begin to execute.

We perform queue wait time prediction and scheduling experiments using four workloads and three scheduling algorithms. The workloads were recorded from an IBM SP at Argonne National Laboratory, an IBM SP at the Cornell Theory Center, and an Intel Paragon at the San Diego Supercomputing Center. The scheduling algorithms are first-come first-served (FCFS), least work first (LWF) and backfill. We find that there is a built-in error when predicting wait times of the LWF algorithm of 34 to 43 percent and a smaller built-in error (3 to 4%) for the backfill algorithm. This error is due to the fact that jobs that have not been enqueued yet can significantly impact the scheduling of applications when using the LWF algorithm. We also find that more accurate run time predictions result in more accurate wait time predictions. Using our run-time prediction technique instead of maximum run times or the run-time prediction techniques of Gibbons [8] or Downey [3] improves run-time prediction error by 39 to 92 percent. This improves wait-time prediction performance by 13 to 87 percent.

Our second problem is to improve the performance of the least work first and backfill scheduling algorithms by using our run-time predictions. These algorithms use run-time predictions when making scheduling decisions and we therefore expect that more accurate run-time predictions will improve scheduling performance. We use the same four workloads to evaluate scheduling performance when using our run-time predictions. We find that the accuracy of the run-time predictions has a minimal effect on the utilization of the systems we are simulating. We also find that using oury of the run-time predictors result in mean wait times that are within 22 percent of the mean wait times that are obtained if the scheduler exactly knows the run times of all of the applications. When comparing the different predictors, our run-time predictor results in 2 to 67 percent smaller mean wait times for the ANL workload, the workload with the highest offered load. No prediction technique clearly outperforms the other techniques when the offered load is low.

The next section summarizes our approach to predicting application run times and the approaches of other researchers. Section 3 describes our queue wait time prediction technique and presents its performance. Section 4 presents the performance of using our run-time predictions in the LWF and backfill scheduling algorithms and Section 5 presents our conclusions.

2 Predicting Application Run Times

In previous work [12] we described our technique for predicting the execution times of parallel applications and compared our technique to those of Downey [3] and Gibbons [8]. This section briefly describes our prediction technique, discusses the scheduling algorithms used with this work, and describes the run-time prediction techniques of the other researchers.

2.1 Our Run-Time Prediction Technique

Our general 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 [12, 3, 5, 8] that similar applications are more likely to have similar run times than applications that have nothing in common. We address the issues of how to define similar and how to generate predictions from similar past applications.

2.1.1 Defining Similarity

A difficulty in developing prediction techniques based on similarity is that two jobs can be compared in many ways. For example, we can compare the application name, submitting user name, executable arguments, submission time, and number of nodes requested. In this work, we are restricted to those values recorded in workload traces obtained from various supercomputer centers. However, because the techniques that we propose are based on the automatic discovery of efficient similarity criteria, we believe that they will apply even if quite different information is available.

The workload traces that we consider are described in Table 1; they originate from Argonne National Laboratory (ANL), the Cornell Theory Center (CTC), and the San Diego Supercomputer Center (SDSC). Table 2 summarizes the information provided in these traces. Text in a field indicates that a particular trace contains the information in question; in the case of "Type," "Queue," or "Class" the text specifies the categories in question.

The general approach to defining similarity taken by ourselves, Downey, and Gibbons is to use characteristics such as those presented in Table 2 to define *templates* that identify a set of *categories* to which jobs can be assigned. For example, the template `(q,u)` specifies that jobs 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 using discrete characteristics 1–6 of Table 2 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)`.

Once a set of templates has been defined (using a search process described later), we can categorize a set of applications (e.g., the workloads of Table 1) by assigning each application to those categories that match its characteristics. Categories need not be disjoint, and hence the same job can occur in several categories. If two jobs fall into the same category, they are judged similar; those that do not coincide in any category are judged dissimilar.

2.1.2 Generating Predictions

We now consider the question of how we generate run-time predictions. The input to this process is a set of templates and a workload for which run-time predictions are required. In addition to the characteristics described in the preceding section, a running time, maximum history, type of data to store, and prediction type are also defined for each template. The running time is how long an application has been running when a prediction is made. Sim-

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

Table 1: Characteristics of the trace data used in our studies.

Workload Name	System	Number of Nodes	Location	When	Number of Requests	Mean Run Time (minutes)
ANL [1]	IBM SP2	120	ANL	3 months of 1996	7994	97.75
CTC	IBM SP2	512	CTC	2 months of 1996	13217	171.14
SDSC95	Intel Paragon	400	SDSC	12 months of 1995	22885	108.21
SDSC96	Intel Paragon	400	SDSC	12 months of 1996	22337	166.98

Table 2: Characteristics recorded in workloads. The column "Abbr" indicates abbreviations used in subsequent discussion.

	Abbr	Characteristic	Argonne	Cornell	SDSC
1	t	Type	batch, interactive	serial, parallel, pvm3	
2	q	Queue			29 to 35 queues
3	c	Class		DSI/PIOFS	
4	u	User	Y	Y	Y
5	s	Loadleveler script		Y	
6	e	Executable	Y		
7	a	Arguments	Y		
8	na	Network adaptor		Y	
9	n	Number of nodes	Y	Y	Y
10		Maximum run time	Y	Y	
11		Submission time	Y	Y	Y
12		Start time	Y	Y	Y
13		Run time	Y	Y	Y

ilarly to the number of nodes, we specify a range size for running time. 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 maximum run times provided by the ANL and CTC workloads). 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, or a logarithmic regression [13, 4]. We found that the mean is the single best predictor so this work only uses means to form predictions.

The output from this process is a set of run-time predictions and associated confidence intervals. (A confidence interval is an interval centered on the run-time prediction within which the actual run time is expected to appear some specified percentage of the time.) The basic algorithm is described below and comprises three phases: initialization, prediction, and incorporation of historical information.

1. Define T, the set of templates to be used, and initialize C, the (initially empty) set of categories.

2. At the time each application a begins to execute:

    (a) Apply the templates in T to the characteristics of a to identify the categories C_a into which the application may fall.

    (b) Eliminate from C_a all categories that are not in C or that cannot provide a valid prediction (i.e., do not have enough data points).

    (c) For each category remaining in C_a, compute a run-time estimate and a confidence interval for the estimate.

    (d) If C_a is not empty, select the estimate with the smallest confidence interval as the run-time prediction for the application.

3. At the time each application a completes execution:

    (a) Identify the set C_a of categories into which the application falls. These categories may or may not exist in C.

    (b) For each category $c_i \in C_a$

        i. If $c_i \notin C$, create c_i in C.

        ii. If $|c_i|$ = maximum history(c_i), remove the oldest point in c_i.

        iii. Insert a into c_i.

Note that steps 2 and 3 operate asynchronously, since historical information for a job cannot be incorporated until the job finishes. Hence, our algorithm suffers from an initial ramp-up phase during which there is insufficient information in C to make predictions. This deficiency could be corrected by using a training set to initialize C.

The use of maximum histories in step 3(b) of our algorithm allows us to control the amount of historical information used when making predictions and the amount of storage space needed to store historical information. A small maximum history means that less historical information is stored, and hence only more recent events are used to make predictions.

2.1.3 Template Definition and Search

We use search techniques to identify good templates for a particular workload; this is in contrast to Gibbons and Downey who use a fixed set of templates. While the number of application characteristics included in our traces is relatively small, the fact that effective template sets may contain many templates means that an exhaustive search is impractical. Our previous work compared greedy and genetic algorithm searches and found that genetic algorithm searches outperform greedy searches. Therefore, we only use genetic algorithm searches in this work.

Genetic algorithms are a probabilistic technique for exploring large search spaces, in which the concept of crossover from biology is used to improve efficiency relative to purely random search [10]. A genetic algorithm evolves individuals over a series of generations. 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 condition we use is that a fixed number of generations have been processed. There are many different variations to this process, and we will next describe the variations we used.

Our individuals represent template sets. Each template set consists of between 1 and 10 templates, and we encode the following information in binary form for each template:

1. Whether a mean or one of the three regressions is used to produce a prediction.

2. Whether absolute or relative run times are used.

3. Whether each of the binary characteristics associated with the workload in question is enabled.

4. Whether node information should be used and, if so, the range size from 1 to 512 in powers of 2.

5. Whether the running time of an application should be used and, if so, the range from 64 to 65536 seconds in powers of 2.

6. Whether the amount of history stored in each category should be limited and, if so, the limit between 2 and 65536 in powers of 2.

A fitness function is used to compute the fitness of each individual and therefore its chance to reproduce. The fitness function should be selected so that the most desirable individuals have higher fitness and produce more offspring, but the diversity of the population must be maintained by not giving the best individuals overwhelming representation in succeeding generations. In our genetic algorithm, we wish to minimize the prediction error and maintain a range of individual fitnesses regardless of whether the range in errors is large or small. The fitness function we use to accomplish this goal is

$$F_{min} + \frac{E_{max} - E}{E_{max} - E_{min}} \times (F_{max} - F_{min}),$$

where E is the error of the individual (template set), E_{min} and E_{max} are the minimum and maximum errors of individuals in the generation, and F_{min} and F_{max} are the desired minimum and maximum fitnesses desired. We chose $F_{max} = 4F_{min}$.

We use a common technique called stochiastic sampling with replacement to select which individuals will mate to produce the next generation. In this technique, each parent is selected from the individuals by selecting individual i with probability $\frac{F_i}{\sum F}$.

The mating or crossover process is accomplished by randomly selecting pairs of individuals to mate and replacing each pair by their children in the new population. The crossover of two individuals proceeds in a slightly nonstandard way because our chromosomes are not fixed length but a multiple of the number of bits used to represent each template. Two children are produced from each crossover by randomly selecting a template i and a position p in the template from the first individual $T_1 = t_{1,1}, \ldots, t_{1,n}$ and randomly selecting a template j in the second individual $T_2 = t_{2,1}, \ldots, t_{2,m}$ so that the resulting individuals will not have more than 10 templates. The new individuals are then $\tilde{T}_1 = t_{1,1}, \ldots, t_{1,i-1}, n_1, t_{2,j+1}, \ldots, t_{2,m}$ and $\tilde{T}_2 = t_{2,1} \ldots t_{2,j-1}, n_2, t_{1,i+1}, \ldots, t_{i,n}$. If there are b bits used to represent each template, n_1 is the first p bits of $t_{1,i}$ concatenated with the last $b - p$ bits of $t_{2,j}$, and n_2 is the first p bits of $t_{2,j}$ concatenated with the last $b - p$ bits of $t_{1,i}$.

In addition to using crossover to produce the individuals of the next generation, we also use a process called elitism whereby the best individuals in each generation survive unmutated to the next generation. We use crossover to produce all but 2 individuals for each new generation and use elitism to select the last 2 individuals for each new generation. The individuals resulting from the crossover process are mutated to help maintain a diversity in the population. Each bit representing the individuals is flipped with a probability of 0.01.

2.1.4 Run-Time Prediction Experiments

We wish to use run time predictions to predict queue wait times and improve the performance of scheduling algorithms. Therefore, we need to determine what workloads to search over to find the best template sets to use. We have already described four sets of trace data that were recorded from supercomputers. Next, we will describe the three scheduling algorithms we consider.

We use the first-come first-served (FCFS), least work first (LWF), and backfill scheduling algorithms in this work. 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 wallclock execution time) the application will perform.

The 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 to make sure that applications behind it in the queue do not delay it; the application may actually start before the reservation time.

Each scheduling algorithm predicts application run times at different times when predicting queue wait times for the jobs in each trace. To try to find the optimal template set to use to predict execution times, we use a workload for each algorithm/trace pair and search over each of these 12 workloads separately. When predicting queue wait times, we predict the wait time of an application when it is submitted. A wait-time prediction in this case requires run-time predictions of all applications in the system so the run-time prediction workload contains predictions for all running and queued jobs every time an application is submitted. We insert data points for an application into our historical database as soon as each application completes.

When using run-time predictions while scheduling, run-time predictions are once again made at different times for each algorithm/trace pair and we attempt to find the optimal template sets to use for each pair. Only the LWF and backfill scheduling algorithms use run-time predictions when making scheduling predictions. The FCFS algorithm does not, so there is no possibility to improve the performance of this algorithm by using more accurate run-time predictions. For the LWF algorithm, all waiting applications are predicted whenever the scheduling algorithm attempts to start an application (when any application is enqueued or finishes). This occurs because the LWF algorithm needs to find the waiting application that will use the least work. For the backfill algorithm, run-time predictions are made for all running and waiting applications whenever the scheduling algorithm attempts to start an application (whenever an application is enqueued or finishes).

We generate our run-time prediction workloads by using maximum run times as run-time predictions during scheduling and generate the prediction events described in the last paragraph. Insertion events are once again generated whenever an application completes. One difficulty is that the LWF and backfill scheduling algorithms use run-time predictions when making scheduling decisions. Therefore, the predictions and insertions made when using maximum run times as the run time predictions will be slightly different than those if a different run-time predictor is used. These run-time prediction workloads should be representative of the predictions and insertions that will be made when scheduling using other run-time predictors.

2.2 Related Work

Gibbons [8, 9] also uses historical information to predict the run times of parallel applications. His technique differs from ours principally in that he uses a fixed set of templates and different characteristics to define templates. He uses the six templates/predictor combinations listed in Table 3. The running time (`rtime`) characteristic indicates how long an application has been executing when a prediction is made for the application. Gibbons produces predictions by examining categories derived from the templates listed in Table 3, in the order listed, until a category that can provide a valid prediction is found. This prediction is then used as the run-time prediction.

The set of templates listed in Table 3 results because Gibbons uses templates of `(u,e)`, `(e)`, and `()` with subtemplates in each template. The subtemplates add the characteristics `n` and `rtime`. Gibbons also uses the requested number of nodes slightly differently from the way we do: rather than having equal-sized ranges specified by a parameter, as we do, he defines the fixed set of exponential ranges 1, 2-3, 4-7, 8-15, and so on.

Table 3: Templates used by Gibbons for run-time prediction.

Number	Template	Predictor
1	`(u,e,n,rtime)`	mean
2	`(u,e)`	linear regression
3	`(e,n,rtime)`	mean
4	`(e)`	linear regression
5	`(n,rtime)`	mean
6	`()`	linear regression

Another difference between Gibbons's technique and ours is how he performs a linear regression on the data in the categories `(u,e)`, `(e)`, and `()`. These categories are used only if one of their subcategories cannot provide a valid prediction. A weighted linear regression is performed on the mean number of nodes and the mean run time of each subcategory that contains data, with each pair weighted by the inverse of the variance of the run times in their subcategory.

Downey [3] uses a different technique to predict the execution time of parallel applications. His procedure is to categorize all applications in the workload, then model the cumulative distribution functions of the run times in each category, and finally use these functions to predict application run times. Downey categorizes applications using the queues that applications are submitted to, although he does state that other characteristics can be used in this categorization. In fact, Downey's prediction technique within a category can be used with our technique for finding the best characteristics to use to categorize applications.

Downey observed that the cumulative distributions of the execution times of the jobs in the workloads he examined can be modeled relatively accurately by using a logarithmic function: $\beta_0 + \beta_1 \ln t$. Once the distribution functions are calculated, he uses two different techniques to produce a run-time prediction. The first technique uses the median lifetime given that an application has executed for a time units. If one assumes the logarithmic model for the cumulative distribution, this equation is

$$\sqrt{a e^{\frac{1.0-\beta_0}{\beta_1}}}.$$

The second technique uses the conditional average lifetime

$$\frac{t_{max} - a}{\log t_{max} - \log a}$$

with $t_{max} = e^{(1.0-\beta_0)/\beta_1}$.

3 Predicting Queue Wait Times

We use the run time predictions described in the previous section to predict queue wait times. Our technique is to perform a scheduling simulation using the predicted run times as the run times of the applications. This will then provide predictions of when applications will start to execute. We simulate the FCFS, LWF, and backfill scheduling algorithms and predict the wait time for each application when the application is submitted to the scheduler. The accuracy of using various run-time predictors is shown in Table 4 through Table 9.

Table 4 shows the wait-time prediction performance when actual run times are used during prediction. No data is shown for the FCFS algorithm because there is no error when computing wait-time predictors in this case. There is no error 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 yet can affect when the jobs currently in the queue can run. This effect larger for the LWF results where if later arriving jobs wish to perform smaller amounts of work, they move to the head of the queue. As you can see in the table, the wait-time prediction error for the LWF algorithm 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. There is also a small error (3-4%) when predicting the wait times for the backfill scheduling algorithm. Any error for the backfill algorithm seems unexpected at first, but errors in wait-time prediction can occur because scheduling is performed using maximum run times. A job J_2 could arrive in the queue, start ahead of an already queued job J_1 because the scheduler does not believe the job J_1 can use those nodes, a running job could finish unexpectedly early, and the job J_1 could have started except that the job J_2 is using nodes that are needed. This example results in a wait-time prediction error for the job J_1 before the job J_2 arrives in the queue.

Table 5 shows the wait-time prediction errors while using maximum run times as run-time predictions. The wait-time prediction error when using actual run times as run-time predictors is 59 to 99 better than the wait-time prediction error of the LWF and Backfill runs when using maximum run times as the run-time predictor. Maximum run times are used to predict run times in scheduling systems such as EASY [11]. These predictions are provided in the ANL and CTC workload and are implied in the SDSC workloads because each of the queues in the two SDSC workload has maximum limits on resource usage. To derive maximum run times for the SDSC workloads, we find the longest running job in each queue and use that as the maximum run time for all jobs in that queue. The maximum run times are provided explicitly or implicitly in the workloads so they are available for use as run-time predictors and can be considered as an upper bound on run-time prediction performance.

Table 6 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 wait-time prediction errors that are from 34 to 77 percent of mean wait times. The best wait-time prediction performance occurs for the ANL workload and the worst for the SDSC96 workload. This is the opposite of what we expect from the run-time prediction errors. The most accurate run-time predictions are for the SDSC96 workload. This implies that accurate run-time predictions are not the only factor that determines the accuracy of wait-time predictions.

The results when using our run-time predictor also show that the mean wait time prediction error is 19 to 42 percent worse than when predicting wait times for the LWF algorithm using actual run times. Finally, using our run-time predictor results in 53 to 86 percent better wait time predictions than when using maximum run times as the run-time predictors.

Table 7 shows the wait-time prediction error when using Gibbons' run time predictor. Our run-time prediction error is 39% to 68% better than Gibbons' and our wait-time prediction errors are 13% to 83% better than Gibbons'. Table 8 and Table 9 shows the wait-time prediction error when using Downey's conditional average and conditional median predictors. The wait-time prediction errors we achieve when using our run-time predictor are 19% to 87% better than these errors and our run-time prediction error is 42% to 92% better. These result and the previous results show that our run-time predictor is more accurate than maximum run times, Gibbons' predictor, or Downey's predictors. The results also show that the wait-time prediction errors are smaller when our run-time predictor is used. This shows that there is a correlation between wait-time prediction error and run-time prediction error.

4 Improving Scheduler Performance

Our second application of run-time predictions is to improve the performance of the LWF and Backfill scheduling algorithms. Table 10 shows the performance of the scheduling algorithms when the actual run times are used as run-time predictors. This is the best performance we can expect in each case and serves as an upper bound on scheduling performance.

Table 11 shows the performance of using maximum run times as run time predictions in terms of average utilization and mean wait time. The scheduling performance

Table 4: Wait-time prediction performance using actual run times.

		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Wait Time
ANL	LWF	37.14	43
ANL	Backfill	5.84	3
CTC	LWF	4.05	39
CTC	Backfill	2.62	10
SDSC95	LWF	5.83	39
SDSC95	Backfill	1.12	4
SDSC96	LWF	3.32	42
SDSC96	Backfill	0.30	3

Table 5: Wait-time prediction performance using maximum run times.

		Run-Time Prediction		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Mean Error (minutes)	Percent of Mean Wait Time
ANL	FCFS	99.23	102	996.67	186
ANL	LWF	203.53	208	97.12	112
ANL	Backfill	134.82	138	429.05	242
CTC	FCFS	243.97	143	125.36	128
CTC	LWF	254.57	149	9.86	94
CTC	Backfill	264.36	154	51.16	190
SDSC95	FCFS	393.31	363	162.72	295
SDSC95	LWF	356.50	329	28.56	191
SDSC95	Backfill	377.50	349	93.81	333
SDSC96	FCFS	394.66	236	47.83	288
SDSC96	LWF	397.30	238	14.19	180
SDSC96	Backfill	397.58	238	39.66	350

Table 6: Wait-time prediction performance using our run time predictor.

		Run-Time Prediction		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Mean Error (minutes)	Percent of Mean Wait Time
ANL	FCFS	44.25	45	182.42	34
ANL	LWF	58.18	60	45.77	53
ANL	Backfill	48.88	50	76.59	43
CTC	FCFS	127.09	74	36.60	37
CTC	LWF	145.30	85	5.58	53
CTC	Backfill	147.97	86	11.85	44
SDSC95	FCFS	54.77	51	30.70	56
SDSC95	LWF	58.18	54	10.10	68
SDSC95	Backfill	59.18	55	13.37	47
SDSC96	FCFS	57.69	35	10.75	65
SDSC96	LWF	56.97	34	5.01	64
SDSC96	Backfill	54.72	33	8.76	77

Table 7: Wait time prediction performance using Gibbons' run time predictor.

		Run-Time Prediction		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Mean Error (minutes)	Percent of Mean Wait Time
ANL	FCFS	93.28	95	350.86	66
ANL	LWF	180.26	184	76.23	91
ANL	Backfill	130.75	134	94.01	53
CTC	FCFS	207.03	121	81.45	83
CTC	LWF	245.92	144	32.34	309
CTC	Backfill	242.25	142	13.57	50
SDSC95	FCFS	114.43	106	54.37	99
SDSC95	LWF	127.48	118	11.60	78
SDSC95	Backfill	123.13	114	20.27	72
SDSC96	FCFS	157.93	95	22.36	135
SDSC96	LWF	158.80	95	6.88	87
SDSC96	Backfill	159.21	95	17.31	153

Table 8: Wait time prediction performance using Downey's conditional average run time predictor.

		Run-Time Prediction		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Mean Error (minutes)	Percent of Mean Wait Time
ANL	FCFS	126.91	130	443.45	83
ANL	LWF	253.25	259	232.24	277
ANL	Backfill	178.34	182	339.10	191
CTC	FCFS	224.88	131	65.22	66
CTC	LWF	255.56	149	14.78	141
CTC	Backfill	253.00	148	17.22	64
SDSC95	FCFS	546.89	505	187.73	340
SDSC95	LWF	688.62	636	35.84	240
SDSC95	Backfill	629.65	582	62.96	223
SDSC96	FCFS	439.33	263	83.62	503
SDSC96	LWF	453.19	271	28.42	361
SDSC96	Backfill	446.24	267	47.11	415

Table 9: Wait time prediction performance using Downey's conditional median run time predictor.

		Run-Time Prediction		Wait-Time Prediction	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Mean Error (minutes)	Percent of Mean Wait Time
ANL	FCFS	97.75	100	534.71	100
ANL	LWF	251.67	257	254.91	304
ANL	Backfill	170.91	175	410.57	232
CTC	FCFS	219.93	129	83.33	85
CTC	LWF	260.71	152	15.47	148
CTC	Backfill	258.77	151	19.35	72
SDSC95	FCFS	286.84	265	62.67	114
SDSC95	LWF	363.78	336	18.28	122
SDSC95	Backfill	331.95	307	27.52	98
SDSC96	FCFS	259.48	155	34.23	206
SDSC96	LWF	265.44	159	12.65	161
SDSC96	Backfill	262.57	157	20.70	183

Table 10: Scheduling performance using actual run times.

		Scheduling	
Workload	Scheduling Algorithm	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	70.34	61.20
ANL	Backfill	71.04	142.45
CTC	LWF	51.28	11.15
CTC	Backfill	51.28	23.75
SDSC95	LWF	41.14	14.48
SDSC95	Backfill	41.14	21.98
SDSC96	LWF	46.79	6.80
SDSC96	Backfill	46.79	10.42

when using the maximum run times can once again be considered an upper bound for comparison. When comparing this data to the data in Table 10, you can see that the maximum run times are an inaccurate predictor but this does not affect the utilization of the simulated parallel computers. Predicting run times with actual run-times when scheduling results in 3 percent to 27 percent lower mean wait times, except in one case where using maximum run times results in 6 percent lower mean wait times. The effect of accurate run-time predictions is highest for the ANL workload which has the largest offered load.

Table 12 shows the performance of using our run-time prediction technique when scheduling. The run-time prediction error in this case is 23% to 93% of mean run times, slightly worse than the results when predicting run-times for wait-time prediction. This worse performance is due to more predictions being performed. First, more predictions are made of applications before they begin executing. These predictions do not have information about how long an application has run which improves prediction performance. Second, more predictions are made of long-running applications, the applications that contribute the largest errors to the mean errors.

Our run-time prediction technique results in mean wait times that are 5 percent better to 4 percent worse than when using actual run times as predictions for the least work first algorithm. For the backfill algorithm, mean wait times when using our run-time predictor our 11 to 22 percent worse. The above result can be understood by noticing that the backfill algorithm requires more accurate run-time predictions than LWF. LWF just needs to know if applications are "big" or "small" and small errors do not greatly affect performance. The performance of the backfill algorithm depends on accurate run-time predictions because it tries to fit applications into time/space slots.

When comparing our run-time prediction technique to using maximum run times, our technique has a minimal effect on the utilization of the systems but it does decrease the mean wait time in 6 of the 8 experiments. Table 13 through Table 15 show the performance of the scheduling algorithms when using Gibbons' and Downey's run-time predictors. The results also indicate that once again, using our run-time predictor does not produce greater utilizations. The results also show that our run-time predictor results in 13 to 50 percent lower mean wait times for the ANL workload, but there is no clearly better run-time predictor for the other three workloads. The ANL workload has much larger mean wait times and higher utilizations (greater offered load) than the other workloads (particularly the SDSC workloads). This may indicate that greater prediction accuracy of our technique when scheduling becomes "hard". To test this hypothesis, we compressed the interrival time of applications by a factor of two for both SDSC workloads and then simulated these two new workloads. We found that our run time predictor results in mean wait times that are 8 percent better on average, but are 43 percent lower to 31 percent higher than mean wait times than obtained when using Gibbons' or Downey's techniques.

The results also show that Downey's conditional average is the worst predictor and Gibbons' predictor is the most accurate. The results also show that our run-time predictor is between 2 and 86 percent better than the other predictors, except for the CTC workload. For this workload, our predictor is the worst. This may be explained by the limited template searches we performed for that workload due to time constraints. The accuracy of the run-time predictions for the CTC workload carriers over to the to the mean wait times of the scheduling algorithms when using the various run-time predictors: our mean wait times are the worst.

5 Conclusions

In this work, we apply predictions of application run times to two separate scheduling problems. The problems are predicting how long applications will wait in queues before executing and improving the performance of scheduling algorithms. Our technique for predicting application run times is to derive a prediction for an application from the run times of previous applications judged similar by a template of key job characteristics. The novelty of our approach lies in the use of search techniques to find the best templates. For the workloads considered in this work, our searches found templates that result in run-time prediction errors that are significantly better than those of other researchers or using user-supplied maximum run times.

We predict queue wait times by using run-time predictions and the algorithms used by schedulers. These two factors are used to simulate scheduling algorithms and decide when applications will execute. Estimates of queue wait times are useful to guide resource selection when several systems are available, to co-allocate resources from multiple systems, to schedule other activities, etc. This technique results in a wait-time prediction error of 34% to 77% of mean wait times when using our run-time predictors. This error is significantly better than when using the run-time predictors of Gibbons, Downey or user-supplied maximum run times. We also find that even if we predict application run times with no error, the wait-time prediction error for the least work first algorithm is significant (34 to 43 percent of mean wait times).

We improve the performance of the least work first and backfill scheduling algorithms by using our run-time predictions when scheduling. We find that the utilization of the parallel computers we simulate does not vary greatly when using different run-time predictors. We also find

Table 11: Scheduling performance using maximum run times as the run time predictor.

		Run-Time Prediction		Scheduling	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	104.12	107	70.70	83.81
ANL	Backfill	154.86	158	71.04	177.14
CTC	LWF	319.82	187	51.28	10.48
CTC	Backfill	75.92	44	51.28	26.86
SDSC95	LWF	411.47	380	41.14	14.95
SDSC95	Backfill	377.50	349	41.14	28.20
SDSC96	LWF	397.30	238	46.79	7.88
SDSC96	Backfill	387.64	232	46.79	11.34

Table 12: Scheduling performance using our run time prediction technique.

		Run-Time Prediction		Scheduling	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	60.97	62	70.28	78.22
ANL	Backfill	50.78	52	71.04	148.77
CTC	LWF	160.13	98	51.28	13.40
CTC	Backfill	163.61	96	51.28	22.54
SDSC95	LWF	70.69	65	41.14	16.19
SDSC95	Backfill	88.68	82	41.14	22.17
SDSC96	LWF	72.58	43	46.79	7.79
SDSC96	Backfill	37.82	23	46.79	10.10

Table 13: Scheduling performance using Gibbons' run time prediction technique.

		Run-Time Prediction		Scheduling	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	92.81	95	70.72	90.36
ANL	Backfill	86.81	89	71.04	181.38
CTC	LWF	158.39	93	51.28	11.04
CTC	Backfill	63.17	34	51.28	27.31
SDSC95	LWF	132.45	122	41.14	15.99
SDSC95	Backfill	91.35	84	41.14	24.83
SDSC96	LWF	178.12	107	46.79	7.51
SDSC96	Backfill	47.56	28	46.79	10.82

Table 14: Scheduling performance using Downey's conditional average run time predictor.

		Run-Time Prediction		Scheduling	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	74.54	76	71.04	154.76
ANL	Backfill	144.80	148	70.88	246.40
CTC	LWF	167.32	98	51.28	9.87
CTC	Backfill	36.19	21	51.28	14.45
SDSC95	LWF	279.04	258	41.14	16.22
SDSC95	Backfill	648.84	709	41.14	20.37
SDSC96	LWF	278.46	167	46.79	7.88
SDSC96	Backfill	470.91	282	46.79	8.25

Table 15: Scheduling performance using Downey's conditional median run time predictor.

		Run-Time Prediction		Scheduling	
Workload	Scheduling Algorithm	Mean Error (minutes)	Percent of Mean Run Time	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	62.22	64	71.04	154.76
ANL	Backfill	144.11	147	71.04	207.17
CTC	LWF	146.85	86	51.28	11.54
CTC	Backfill	30.06	18	51.28	16.72
SDSC95	LWF	127.97	118	41.14	16.36
SDSC95	Backfill	347.42	321	41.14	19.56
SDSC96	LWF	138.44	83	46.79	7.80
SDSC96	Backfill	275.74	165	46.79	8.02

that using our run-time predictions does improve the mean wait times in general. In particular, our more accurate run-time predictors have the largest impact on mean wait time for the ANL workload, which has the highest utilization. In this workload, the mean wait times are 7% to 67% lower when using our run-time predictions than when using other run-time predictions. We also find that on average, the mean wait time when using our predictor is within 8 percent of the mean wait time that would occur if the scheduler knows the exact run times of the applications. The mean wait time when using our technique ranges from 5 percent better to 22 percent worse than when scheduling with the actual run times.

In future work, we will investigate an alternative method for predicting queue wait times. This method will use the current state of the scheduling system (number of applications in each queue, time of day, etc.) and historical information on queue wait times during similar past states to predict queue wait times. We hope this technique will improve wait-time prediction error, particularly for the LWF algorithm that has a large built-in error using the technique presented here. Further, we will expand our work in using run-time prediction techniques for scheduling to the problem of combining queue based scheduling and reservations. Reservations are one way to co-allocate resources in metacomputing systems [1, 6, 2, 7]. Support for resource co-allocation is crucial to large-scale applications that require resources from more than one parallel computer.

References

[1] C. Catlett and L. Smarr. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[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.

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

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

[5] Dror Feitelson and Bill Nitzberg. Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860. *Lecture Notes on Computer Science*, 949:337–360, 1995.

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

[7] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann, 1999.

[8] Richard Gibbons. A Historical Application Profiler for Use by Parallel Scheculers. *Lecture Notes on Computer Science*, 1297:58–75, 1997.

[9] Richard Gibbons. A Historical Profiler for Use by Parallel Schedulers. Master's thesis, University of Toronto, 1997.

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

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

[12] Warren Smith, Ian Foster, and Valerie Taylor. Predicting Application Run Times Using Historical Information. *Lecture Notes on Computer Science*, 1459:122–142, 1998.

[13] Neil Weiss and Matthew Hassett. *Introductory Statistics*. Addison-Wesley, 1982.