Predicting Application Run Times Using Historical Information

Warren Smith *† Ian Foster * Valerie Taylor †

{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

We present a technique for deriving predictions for the run times of parallel applications from the run times of "similar" applications that have executed in the past. The novel aspect of our work is the use of search techniques to determine those application characteristics that yield the best definition of similarity for the purpose of making predictions. We use four workloads recorded from parallel computers at Argonne National Laboratory, the Cornell Theory Center, and the San Diego Supercomputer Center to evaluate the effectiveness of our approach. We show that on these workloads our techniques achieve predictions that are between 14 and 60 percent better than those achieved by other researchers; our approach achieves mean prediction errors that are between 41 and 65 percent of mean application run times.

1 Introduction

Predictions of application run time can be used to improve the performance of scheduling algorithms [8] and to predict how long a request will wait for resources [4]. We believe that run-time predictions can also be useful in metacomputing environments in several different ways. First, they are useful as a means of estimating queue times and hence guiding selections from among various resources. Second, they are useful when attempting to gain simultaneous access to resources from multiple scheduling systems [2].

The problem of how to generate run time estimates has been examined by Downey [4] and Gibbons [8]. Both adopt the approach of making predictions for future jobs by applying a "template" of job characteristics to identify "similar" jobs that have executed in the past. Unfortunately, their techniques are not very accurate, with errors frequently exceeding execution times.

We believe that the key to making more accurate predictions is to be more careful about which past jobs are used to make predictions. Accordingly, we apply greedy and genetic algorithm search techniques to identify templates that perform well when partitioning jobs into categories within which jobs are judged to be similar. We also examine and evaluate a number of variants of our basic prediction strategy. We look at whether it is useful to use linear regression techniques to exploit node count information when jobs in a category have different node counts. We also look at the effect of varying the amount of past information used to make predictions, and we consider the impact of using user-supplied maximum run times on prediction accuracy.

We evaluate our techniques using four workloads recorded from supercomputer centers. This study shows that the use of search techniques makes a significant difference to prediction accuracy: our prediction algorithm achieves prediction errors that are 14 to 49 percent lower than those achieved by Gibbons, depending on the workload, and 23 to 60 percent lower than those achieved by Downey. The genetic algorithm search performs better than greedy search.

The rest of the paper is structured as follows. Section 2 describes how we define application similarity, perform predictions, and use search techniques to identify good templates. Section 3 describes the results when our algorithm is applied to supercomputer workloads. Section 4 compares our techniques and results with those of other researchers. Section 5 presents our conclusions and notes directions for further work. An appendix provides details of the statistical methods used in our work.

2 Prediction Techniques

Both intuition and previous work [6, 4, 8] indicate that "similar" applications are more likely to have similar run times than applications that have nothing in common. This observation is the basis for our approach to the prediction problem, which is to derive run-time predictions from historical information of previous similar runs.

In order to translate this general approach into a specific prediction method, we need to answer two questions:

1. *How do we define "similar"?* Jobs may be judged similar because they are submitted by the same user, at the same time, on the same computer, with the same arguments, on the same number of nodes, and so on. We require techniques for answering the question: Are these two jobs similar?

2. *How do we generate predictions?* A definition of similarity allows us to partition a set of previously executed jobs into buckets or categories within which all are similar. We can then generate predictions by, for example, computing a simple mean of the run times in a category.

We structure the description of our approach in terms of these two issues.

2.1 Defining Similarity

In previous work, Downey [4] and Gibbons [8] demonstrated the value of using historical run-time information for "similar" jobs to predict run times for the purpose of improving scheduling performance and predicting wait times in queues. However, both Downey and Gibbons restricted themselves to relatively simple definitions of similarity. A major contribution of the present work is to show that more sophisticated definitions of similarity can lead to significant improvements in prediction accuracy.

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. We can conceivably also consider more esoteric parameters such as home directory, files staged, executable size, and account to which the run is charged. 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 characteristics described in rows 1–9 are physical characteristics of the job itself. Characteristic 10, "maximum run time," is information provided by the user and is used by the ANL and CTC schedulers to improve scheduling performance. Rows 11 and 12 are temporal information, which we have not used in our work to date; we hope to evaluate the utility of this information in future work. Characteristic 13 is the run time that we seek to predict.

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 ANL SP, this template generates categories such as `(night,wsmith)`, `(day,wsmith)`, and `(night,foster)`.

We find that using discrete characteristics 1–8 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 (see Section 2.4) we can categorize a set of jobs (e.g., the workloads of Table 1) by assigning each job 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.

2.2 Generating Predictions

We now consider the question of how we generate run-time predictions. The input to this process is a set of templates T and a workload W for which run-time predictions are required. In addition to the characteristics described in the preceding section, a maximum history, type of data, and prediction type are also defined for each 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, denoted by `act`, or a relative run time, denoted by `rel`. A relative run-time incorporates information about user-supplied run time estimates by storing the ratio of the

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

Workload Name	System	Number of Nodes	Location	When	Number of Requests	Mean Run Time (minutes)
ANL	IBM SP2	80	ANL	3 months of 1996	7994	97.40
CTC	IBM SP2	512	CTC	11 months of 1996	79302	182.18
SDSC95	Intel Paragon	400	SDSC	12 months of 1995	22885	107.76
SDSC96	Intel Paragon	400	SDSC	12 months of 1996	22337	166.48

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

	Abbr	Characteristic	Argonne SP	Cornell SP	SDSC Paragon
1	t	Type	batch/interactive	serial/parallel/pvm3	
2	q	Queue	day/night		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

actual run time to the user-supplied estimate (as described in Section 2.3). The prediction type determines how a run-time prediction is made from the data in each category generated from a template. We use a mean, denoted by `mean`, or a linear regression, denoted by `lr`, to compute estimates.

The output from this process is a set of run-time predictions and associated confidence intervals. (As discussed in the appendix, 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 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, as described in the appendix.
   (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 $c_i \notin C$, then create c_i in C.
      ii. If $|c_i| = \text{maximum history}(c_i)$, 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.

We now discuss how a prediction is generated from the contents of a category in step 2(c) of our algorithm. We consider two techniques in this paper. The first simply computes the mean of the run times contained in the category. The second attempts to exploit the additional information provided by the node counts associated with previous run times by performing a linear regression to compute coefficients a and b for the equation $R = aN + b$, where N is node count and R is run time. This equation is then used to make the prediction. The techniques used to compute confidence intervals in these two cases, which we term *mean* and *linear regression* predictors, respectively, are described in the appendix.

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.

2.3 User Guidance

Another approach to obtaining accurate run-time predictions is to ask users for this information at the time of job submission. This approach may be viewed as complementary to the prediction techniques discussed previously, since historical information presumably can be used to evaluate the accuracy of user predictions.

Unfortunately, none of the systems for which we have workload traces ask users to explicitly provide information about expected run times. However, all of the workloads provide implicit user estimates. The ANL and CTC workloads include user-supplied maximum run times. This information is interesting because users have some incentive to provide accurate estimates. The ANL and CTC systems both kill a job after its maximum run time has elapsed, so users have incentive not to underestimate this value. Both systems also use the maximum run time to determine when a job can be fit into a free slot, so users also have incentive not to overestimate this value.

Users also provide implicit estimates of run times in the SDSC workloads. The scheduler for the SDSC Paragon has many different queues with different priorities and different limits on application resource use. When users pick a queue to submit a request to, they are providing a prediction of the resource use of their application. Queues that have lower resource limits tend to have higher priority, and applications in these queues tend to begin executing quickly, so users are motivated to submit to queues with low resource limits. Also, the scheduler will kill applications that go over their resource limits, so users are motivated not to submit to queues with resource limits that are too low.

A simple approach to exploiting user guidance is to base predictions not on the run times of previous applications, but on the relationship between application run times and user predictions. For example, a prediction for the ratio of actual run time to user-predicted run time can be used along with the user-predicted run time of a particular application to predict the run time of the application. We use this technique for the ANL and CTC workloads by storing relative run times, the run times divided by the user-specified maximum run times, as data points in categories instead of the actual run times.

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

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

2.4 Template Definition and Search

We have not yet addressed the question of how we define an appropriate set of templates. This is a nontrivial problem. If too few categories are defined, we group too many unrelated jobs together, and obtain poor predictions. On the other hand, if too many categories are defined, we have too few jobs in a category to make accurate predictions.

Downey and Gibbons both selected a fixed set of templates to use for all of their predictions. Downey uses only a single template containing only the queue name; prediction is based on a conditional probability function. Gibbons uses the six templates/predictor combinations listed in Table 3. The `age` characteristic indicates how long an application has been executing when a prediction is made. Section 4 discusses further details of their approaches and a comparison with our work.

We use search techniques to identify good templates for a particular workload. 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.

Hence, we consider alternative search techniques. Results for greedy and genetic algorithm search are presented in this paper.

The greedy and genetic algorithms both take as input a workload W from Table 1 and produce as output a template set; they differ in the techniques used to explore different template sets. Both algorithms evaluate the effectiveness of a template set T by applying the algorithm of Section 2.2 to workload W. Predicted and actual values are compared to determine for W and T both the mean error and the percentage of predictions that fall within the 90 percent confidence interval.

2.4.1 Greedy Algorithm

The greedy algorithm proceeds iteratively to construct a template set $T = \{t_i\}$ with each t_i of the form

$$\{\ ()\ (h_{1,1})\ (h_{2,1}, h_{2,2}), \ldots, (h_{i,1}, h_{i,2}, \ldots, h_{i,i})\ \},$$

where every $h_{j,k}$ is one of the n characteristics $h_1, h_2 \ldots, h_n$ from which templates can be constructed for the workload in question. The search over workload W is performed with the following algorithm:

1. Set the template set $T = \{()\}$
2. For $i = 1$ to n
   (a) Set T_c to contain the $\binom{n}{i}$ different templates that contain i characteristics.
   (b) For each template t_c in T_c
      i. Create a candidate template set $X_c = T \cup \{t_c\}$
      ii. Apply the algorithm of Section 2.2 to W and X_c, and determine mean error
   (c) Select the T_c with the lowest mean error, and add the associated template t_c to T

Our greedy algorithm can search over any set of characteristics. Here, however, because of time constraints we do not present searches over maximum history sizes. This restriction reduces the size of the search space, but potentially also results in less effective templates.

2.4.2 Genetic Algorithm Search

The second search algorithm that we consider uses genetic algorithm techniques to achieve a more detailed exploration of the search space. 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 [10]. A genetic algorithm evolves individuals over a series of generations. The processing 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 linear regression prediction is performed
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 a maximum history should be used and, if so, the size between 1 and 32,768 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 therefore have 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

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

where E is the error of the individual 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 $F_{max} = 4 \cdot F_{min}$.

We use a common technique called stochiastic sampling without replacement to select which individuals will mate to produce the next generation. In this technique, each individual is selected $\lfloor \frac{F}{F_{avg}} \rfloor$ times to be a parent. The rest of the parents are selected by Bernoulli trials where each individual is selected, in order, with a probability of $F - F_{avg} \lfloor \frac{F}{F_{avg}} \rfloor$ until all parents are selected.

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 in the template p 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 $T_1 = t_{1,1}, \ldots, t_{1,i-1}, n_1, t_{2,j+1}, \ldots, t_{2,m}$ and $T_2 = t_{2,1} \ldots t_{2,j-1}, n_2, t_{1,i+1}, \ldots, ti, n$. 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 $t2, j$. and n_2 is the first p bits of $t_{2,j}$ concatenated with the last $b-p$ bits of $t1, i$.

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 0.001.

3 Experimental Results

In the preceding section we described our basic approach to run-time prediction. We introduced the concept of *template search* as a means of identifying efficient criteria for selecting "similar" jobs in historical workloads. We also noted potential refinements to this basic technique, including the use of alternative search methods (greedy vs. genetic), the introduction of node count information via linear regression, support for user guidance, and the potential for varying the amount of historical information used. In the rest of this paper, we discuss experimental studies that we have performed to evaluate the effectiveness of our techniques and the significant of the refinements just noted.

Our experiments used the workload traces summarized in Table 1 and are intended to answer the following questions:

- How effectively do our greedy and genetic search algorithms perform?
- What is the relative effectiveness of mean and linear regression predictors?
- What is the impact of user guidance as represented by the maximum run times provided on the ANL and CTC SPs?
- What is the impact of varying the number of nodes in each category on prediction performance?
- What are the trends for the best templates in the workloads?
- How do our techniques compare with those of Downey and Gibbons?

3.1 Greedy Search

Figure 1 shows the results of performing a greedy search for the best category templates for all four workloads. Several trends can be observed from this data. First, adding a second template with a single characteristic results in the most dramatic improvement in performance. The addition of this template has the least effect for the CTC workload where performance is improved between 5 and 48 percent and has the greatest effect for the SDSC workloads which improve between 34 and 48 percent. The addition of templates using up to all possible characteristics results in less improvement than the addition of the template containing a single characteristic. The improvements range from an additional 1 to 18 percent improvement with the ANL workload seeing the most benefit and the SDSC96 workload seeing the least.

Second, the graphs show that the mean is a better predictor than linear regression except when a single template is used with the SDSC workloads. The final predictors obtained by using means are between 2 and 33 percent more accurate than those based on linear regressions. The impact of the choice of predictor on accuracy is greatest in the ANL and least in the SDSC96 workload.

A third trend, evident in the ANL and CTC results, is that using the relative run times gives a significant improvement in performance. When this information is incorporated, prediction accuracy increases between 23 and 43 percent with the ANL workload benefiting most.

Table 4 lists for each workload the accuracy of the best category templates found by the greedy search. In the last column, the mean error is expressed as a fraction of mean run time. Mean errors of between 42 and 70 percent of mean run times may appear high; however, as we will see later, these figures are comparable to those achieved by other techniques, and genetic search performs significantly better.

Looking at the templates listed in Table 4, we observe that for the ANL and CTC workloads, the executable and user name are both important characteristics to use when deciding whether applications are similar. Examination of other data gathered during the experiments shows that these two characteristics are highly correlated: substituting `u` for `e` or `s` or vice versa in templates results in similar performance in many experiments. This observation may imply that users tend to run a single application on these parallel computers.

The templates selected for the SDSC workloads indicate that the user who submits an application is more

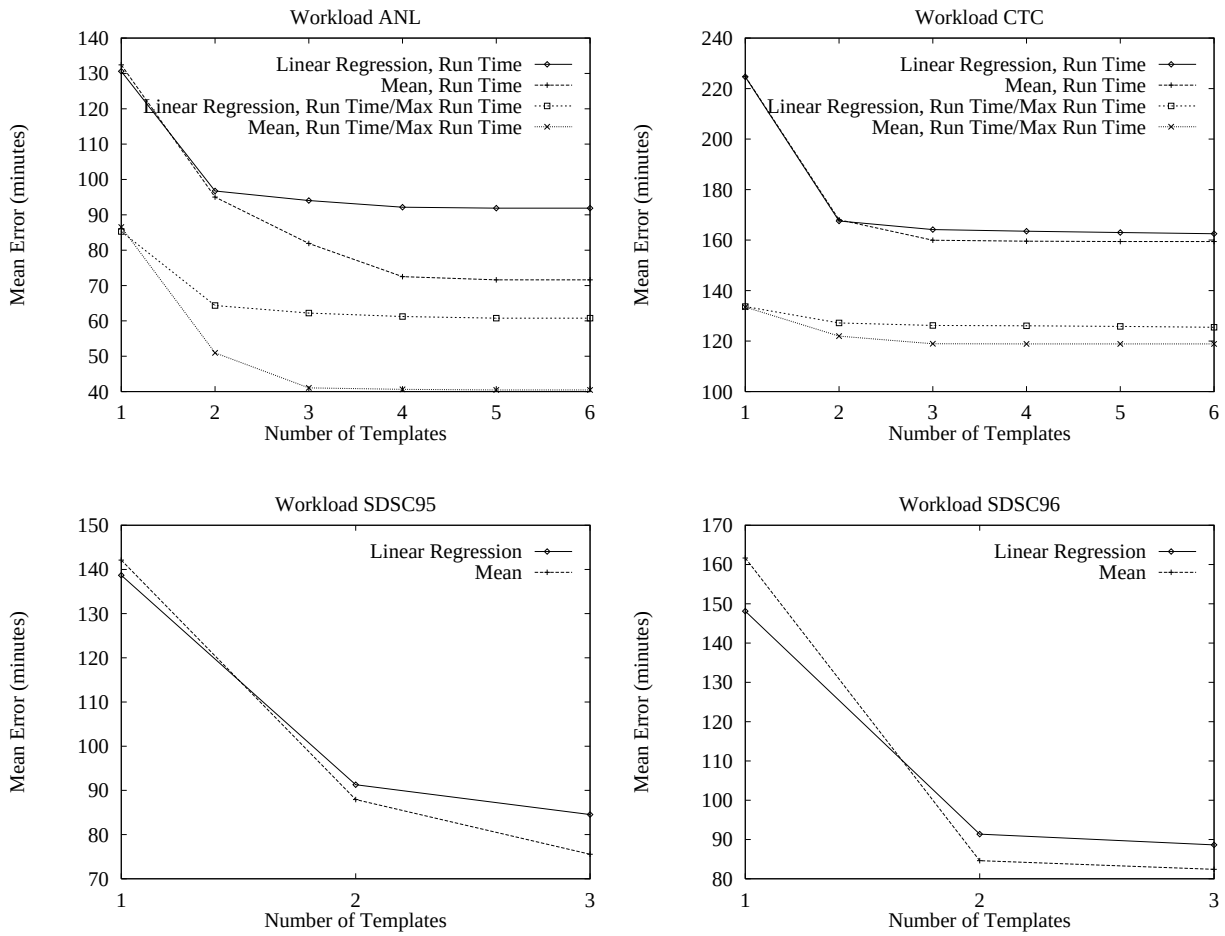

Figure 1: Mean errors of greedy searches

important in determining application similarity than the queue to which an application is submitted. Furthermore, Figure 1 shows that adding the third template results in performance improvements of only 2 to 12 percent on the SDSC95 and SDSC96 workloads. Comparing this result with the greater improvements obtained when relative run times are used in the ANL and CTC workloads suggests that SDSC queue classes are not good user-specified run-time estimates. It would be interesting to use the resource limits associated with queues as maximum run times. However, this information was not available to us when this paper was being written.

Figure 2 shows for the ANL workload the percentage of actual run times that fall within the 90 percent confidence interval. We see that the confidence intervals generated when using a linear regression predictor are not accurate. Similar results are obtained for the other workloads.

We next perform a second series of greedy searches to identify the impact of using node information when defining categories. We use node ranges when defining categories as described in Section 2.1. The results of these searches are shown in Table 5. Because of time constraints, no results are available for the CTC workload.

The table shows that using node information improves prediction performance by 2 and 10 percent with the largest improvement for the San Diego workloads. This information and the fact that characteristics such as executable, user name, and arguments are selected before nodes when searching for templates indicates that the importance of node information to prediction accuracy is only moderate.

Further, the greedy search selects relatively small node range sizes coupled with user name or executable. This fact indicates, as expected, that an application executes for similar times on similar numbers of nodes.

3.2 Genetic Algorithm Search

Figure 3 shows the progress of the genetic algorithm search of the ANL workload. While the average and maximum errors tend to decrease significantly as evolution proceeds, the minimum error decreases only slightly. This behavior suggests that the genetic algorithm is working correctly but that it is not difficult to find individual templates with low prediction errors.

As shown in Table 6, the best templates found during the genetic algorithm search provide mean errors that are 2 to 12 percent less than the best templates found during the greedy search. The largest improvements are obtained on the CTC and SDSC95 workloads. These results indicate that the genetic search performs slightly better than the greedy search. This difference in performance may increase if the search space becomes larger by, for example, including the maximum history characteristic while searching.

The template sets identified by the genetic search procedure are listed in Table 7. Studying these and other template sets produced by genetic search, we see that the mean is not uniformly used as a predictor. From the results of the greedy searches, the mean is clearly a better predictor in general but these results indicate that combining mean and linear regression predictors does provide a performance benefit. Similarly to the greedy searches of the ANL and CTC workloads, using relative run times as data points provides the best performance.

A third observation is that node information is used in the templates of Table 7 and throughout the best templates found during the genetic search. This confirms the observation made during the greedy search that using node information when defining templates results in improved prediction performance.

4 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.

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 use the characteristics `n` and `age` (how long an application has executed). In our work we have used the user, executable, and nodes characteristics. We do not use the age of applications in this discussion, although this characteristic has value [4, 3]. 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.

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.

Table 8 compares the performance of Gibbons's technique with our technique. Using code supplied by Gibbons, we applied his technique to our workloads. We see that our greedy search results in templates that perform

Table 4: Best predictions found during greedy first search.

Workload	Predictor	Data Point	Template Set	Mean Error (minutes)	Percentage of Mean Run Time
ANL	mean	relative / run time	(),(e),(u,a),(t,q,u), (t,q,u,e), (t,q,u,e,a)	40.46	41.54
CTC	mean	relative / run time	(), (u), (u,s), (t,c,s), (t,u,s,ni), (t,c,u,s,ni)	118.89	65.25
SDSC95	mean	run time	(),(u), (q,u)	75.56	70.12
SDSC96	mean	run time	(),(u),(q,u))	82.40	49.50

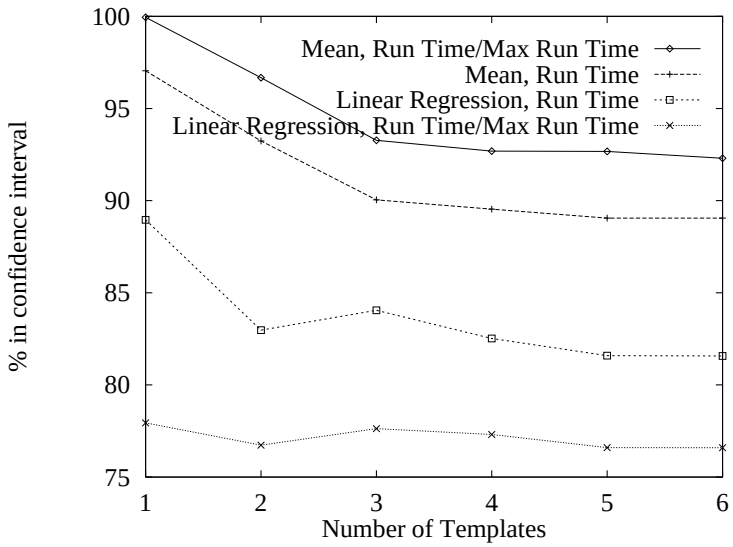

Figure 2: Percentage of actual run times that fall within the 90 percent confidence interval, for workload ANL.

Table 5: Best predictions found during second greedy search.

Workload	Predictor	Data Point	Template Set	Mean Error (minutes)	Percentage of Mean Run Time
ANL	mean	relative run time	(),(e),(u,a),(t,u,n=2), (q,u,e,n=32), (t,u,e,a,n=16), (t,q,u,e,a,n=4)	39.66	40.72
SDSC95	mean	run time	(),(u),(u,n=1),(q,u,n=1)	67.63	62.76
SDSC96	mean	run time	(),(u),(u,n=4),(q,u,n=8)	76.20	45.77

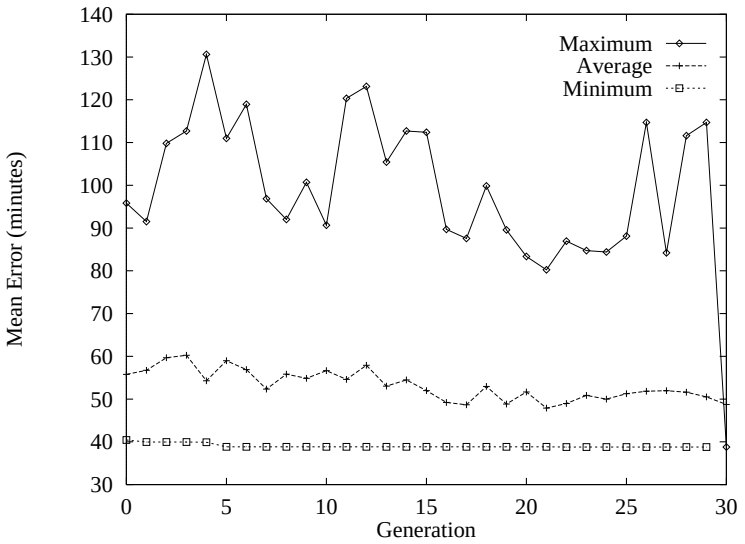

Figure 3: Errors during genetic algorithm search of workload ANL

Table 6: Performance of the best templates found during genetic algorithm search. Results for greedy search are also presented, for comparison.

Workload	Genetic Algorithm		Greedy	
	Mean Error (minutes)	Percentage of Mean Run Time	Mean Error (minutes)	Percentage of Mean Run Time
ANL	38.33	39.35	39.66	40.72
CTC	106.73	58.58	118.89	65.25
SDSC95	59.65	55.35	67.63	62.76
SDSC96	74.56	44.79	76.20	45.77

Table 7: The best templates found during genetic algorithm search

Workload	Best Template Set
ANL	(q,e,a,n=4,mean,rel), (q,u,n=4,lr,rel), (q,u,e,n=32,lr,rel) (q,u,e,a,n=32,mean,rel), (t,u,a,n=4,lr,rel), (t,u,a,n=4,mean,rel) (t,u,e,a,n=64,lr,rel) (t,q,u,e,a,n=128,mean,rel)
CTC	(u,n=512,mean,rel), (c,e,a,ni,n=4,mean,rel)
SDSC95	(q,u,n=1,mean,act), (q,n=16,lr,act) (q,u,n=16,lr,act), (q,u,n=4,lr,act)
SDSC96	(u,n=1,mean,act), (q,n=4,lr,act), (q,u,n=4,lr,act), (q,u,n=128,mean,act), (q,u,n=16,mean,act), (q,u,n=2,mean,act), (q,u,n=4,mean,act)

between 2 percent worse for the SDSC95 workload to 46 percent better than Gibbons's technique. This table also shows that our genetic algorithm search finds template sets that have between 14 and 49 percent lower mean error than the template sets Gibbons selected.

In his original work, Gibbons did not have access to workloads that contained the maximum run time of applications, so he could not use this information to refine his technique. In order to study the potential benefit of this data on his approach, we reran his predictor while using application run time divided by the user-specified maximum run time. Table 9 shows our results. Using maximum run times improves the performance of Gibbons's prediction technique on both workloads, although not to the level of the predictions found during our genetic algorithm search.

Downey [4] uses a different technique to predict the execution time of parallel applications. His technique is to model the applications in a workload and then use these models to predict application run times. 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.

Downey observed that the cumulative distributions can be modeled by using a logarithmic function: $\beta_0 + \beta_1 \ln t$, although this function is not completely accurate for all distributions he observed. 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. Assuming the logorithmic model for the cumulative distribution, this equation is

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

The second technique uses the conditional average lifetime

$$\frac{t_{max} - 1}{\log t_{max} - \log a}$$

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

The performance of both of these techniques are shown in Table 10. We have reimplemented Downey's technique as described in [4] and used his technique on our workloads. The predictions are made assuming that the application being predicted has executed for one second. The data shows that of Downey's two techniques, using the median has better performance in general and the template sets found by our genetic algorithm perform 23 to 60 percent better than the Downey's best predictors. There are two reasons for this performance difference. First, our techniques use more characteristics than just the queue name to determine which applications are similar. Second, calculating a regression to the cumulative distribution functions minimizes the error for jobs of all ages while we concentrate on accurately predicting jobs of age 0.

5 Conclusions

We have described a novel technique for using historical information to predict the run times of parallel applications. Our technique is to derive a prediction for a job from the run times of previous jobs 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. We experimented with the use of both a greedy search and a genetic algorithm search for this purpose, and we found that the genetic search performs better for every workload and finds templates that result in prediction errors of 40 to 60 percent of mean run times in four supercomputer center workloads. The greedy search finds templates that result in prediction errors of 41 to 65 percent of mean run times. Furthermore, these templates provide more accurate run-time estimates than the techniques of other researchers: we achieve mean errors that are 14 to 49 percent lower error than those obtained by Gibbons and 23 to 60 percent lower error than Downey.

We find that using user guidance in the form of user-specified maximum run times when performing predictions results in a significant 23 percent to 43 percent improvement in performance for the Argonne and Cornell workloads. We used both means and linear regressions to produce run-time estimates from similar past applications and found that means provide more accurate predictions in general. For the best templates found in the greedy search, using the mean for predictions resulted in between 2 percent and 33 percent smaller errors. The genetic search shows that combining templates that use both mean and linear regression improves performance.

Our work also provides insights into the job characteristics that are most useful for identifying similar jobs. We find that the names of the submitting user and the application are the most useful and that the number of nodes is also valuable.

In future work, we hope to use search techniques to explore yet more sophisticated prediction techniques. For example, we are interested in understanding whether it is useful to constrain the amount of history information used to make predictions. We are also interested in understanding the potential benefit of using submission time, start time, and application age when making predictions. We may also consider more sophisticated search techniques and more flexible definitions of similarity. For example, instead of applications being either similar or disimilar, there could be a range of similarities. A second direction

Table 8: Comparison of our prediction technique with that of Gibbons

Workload	Gibbons's Mean Error (minutes)	Our Mean Error	
		Greedy Search (minutes)	Genetic Algorithm (minutes)
ANL	75.26	40.46	38.33
CTC	124.06	118.89	106.73
SDSC95	74.05	75.56	59.65
SDSC96	122.55	82.40	74.56

Table 9: Comparison of our prediction technique to that of Gibbons, when Gibbons's technique is modified to use run times divided by maximum run times as data points

Workload	Gibbons's Mean Error (minutes)	Our Mean Error	
		Greedy Search (minutes)	Genetic Algorithm (minutes)
ANL	49.47	40.46	38.33
CTC	107.41	118.89	106.73

Table 10: Comparison of our prediction technique with that of Downey

Workload	Downey's Mean Error		Our Mean Error	
	Conditional Median Lifetime (minutes)	Conditional Average Lifetime (minutes)	Greedy Search (minutes)	Genetic Algorithm (minutes)
ANL	96.90	204.24	40.46	38.33
CTC	180.13	222.14	118.89	106.73
SDSC95	104.80	171.78	75.56	59.65
SDSC96	96.90	102.73	82.40	74.56

for future work is to apply our techniques to the problem of selecting and co-allocating resources in metacomputing systems [1, 7, 2]

Acknowledgments

We thank the Mathematics and Computer Science Division of Argonne National Laboratory, the Cornell Theory Center, and the San Diego Supercomputer Center for providing us with the trace data used in this work. We also thank Gene Rackow for helping to record the ANL trace, Allen Downey for providing the SDSC workloads, Jerry Gerner for providing the CTC workload, and Richard Gibbons for providing us with the code used for the comparative analysis.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and a NSF Young Investigator award under Grant CCR-9215482.

Appendix: Statistical Methods

We use statistical methods [11, 5] to calculate run-time estimates and confidence intervals from categories. A category contains a set of data points called a *sample*, which are a subset of all data points that will be placed in the category, *the population*. We use a sample to produce an estimate using either a mean or a linear regression. This estimate includes a confidence interval that is useful as a measure of the expected accuracy of this prediction. If the $X\%$ confidence interval is of size c, a new data point will be within c units of the prediction $X\%$ of the time. A smaller confidence interval indicates a more accurate prediction.

A mean is simply the sum of the data points divided by the number of data points. A confidence interval is computed for a mean by assuming that the data points in our sample S are an accurate representation of all data points in the population P of data points that will ever be placed in a category. The sample is an accurate representation if they are taken randomly from the population and the sample is large enough. We assume that the sample is random, even though it consists of the run times of a series of applications that have completed in the recent past. If the sample is not large enough, the sample mean $\overline{x}$ will not be nearly equal to the population mean μ, and the sample standard deviation s will not be near to the population standard deviation σ. The prediction and confidence interval we compute will not be accurate in this case. In fact, the central limit theorem states that a sample size of at least 30 is needed for $\overline{x}$ to approximate μ, although the exact sample size needed is dependent on σ and the standard deviation desired for $\overline{x}$ [11].

We used a minimum sample size of 2 when making our predictions in practice. This is because while a small sample size may result in $\overline{x}$ not being nearly equal to μ, we find that an estimate from a category that uses many characteristics but has a small sample is more accurate than an estimate from a category that uses few characteristics but has a larger sample size.

The $X\%$ confidence interval can be computed when using the sample mean as a predictor by applying Chebychev's theorem. This theorem states that the portion of data that lies within k standard deviations to either side of the mean is at least $1 - \frac{1}{k^2}$ for any data set. We need only compute the sample standard deviation and k such that $1 - \frac{1}{k^2} = \frac{X}{100}$.

Our second technique for producing a prediction is to perform a linear regression to a sample using the equation

$$t = b_0 + b_1 n,$$

where n is the number of nodes requested and t is the run time. This type of prediction attempts to use information about the number of nodes requested. A confidence interval can be constructed by observing how close the data points are to this line. The confidence interval is computed by the equation

$$t_{\frac{\alpha}{2}} \sqrt{MSE} \sqrt{1 + \frac{1}{N} + \frac{(n_0 - \bar{n})^2}{\sum n^2 - \frac{(\sum n)^2}{N}}}$$

, where N is the sample size, MSE is the mean squared error of the sample, n_0 is the number of nodes requested for the application being predicted, and $\bar{n}$ is the mean number of nodes in the sample. Alpha is computed with the equation

$$\alpha = 1 - \frac{X\%}{100}$$

if the $X\%$ confidence interval is desired and $t_{\frac{\alpha}{2}}$ is the Student's t-distribution with $n-2$ degrees of freedom [11, 5].

References

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

[2] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metasystems. In *The IPPS'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[3] 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.

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

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

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

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

[8] Richard Gibbons. A Historical Application Profiler for Use by Parallel Scheculers. *Lecture Notes on Computer Science*, pages 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] Neil Weiss and Matthew Hassett. *Introductory Statistics.* Addison-Wesley, 1982.