Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 64 (2004) 1007–1016
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

# Predicting application run times with historical information

Warren Smith[a], Ian Foster[b], Valerie Taylor[c,*]

[a]Computer Sciences Corporation, NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA 94035, USA
[b]Department of Computer Science, University of Chicago, Chicago, IL 60637, USA
[c]Department of Computer Science, Texas A&M University, College Station, Texas 77843, USA

Received 29 September 1998; received in revised form 10 May 2004

**Abstract**

We present a technique for predicting the run times of parallel applications based upon 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 21 and 64 percent better than those achieved by other techniques; our approach achieves mean prediction errors that are between 29 and 59 percent of mean application run times.
© 2004 Published by Elsevier Inc.

## 1. Introduction

Predictions of application run time can be used to improve the performance of scheduling algorithms [7,10–12] and to predict how long a request will wait for resources [3,10–12]. 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 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 supercomputers. This study shows that the use of search techniques makes a significant improvement in prediction accuracy: our prediction algorithm achieves prediction errors that are 21–61 percent lower than those achieved by Gibbons, depending on the workload, and 41–64 percent lower than those achieved by Downey. The templates found by the genetic algorithm search outperform the templates found by the 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 [3,4,7] indicate that "similar" applications are more likely to have similar run

* Corresponding author. Fax: +1-979-845-1420.
*E-mail addresses:* wwsmith@arc.nasa.gov (W. Smith), foster@mcs.anl.gov (I. Foster), taylor@cs.tamu.edu (V. Taylor).

0743-7315/$ - see front matter © 2004 Published by Elsevier Inc.
doi:10.1016/j.jpdc.2004.06.008

Table 1
Characteristics of the workloads used in our studies

| Workload name | System | Number of nodes | Location | When | Number of requests | Mean run time (min) |
|---|---|---|---|---|---|---|
| ANL[a] | 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 |

[a]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.

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.

To translate this general approach into a specific prediction method, we need to address two issues: (1) how to define “similar” and (2) how to generate predictions. These issues are addressed in the following sections.

## 2.1. Defining similarity

In previous work, Downey [3] and Gibbons [7] demonstrated the value of using historical run-time information to identify “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 fixed definitions of similarity. A major contribution of the present work is to show that searching for the most appropriate definitions of similarity can lead to significant improvements in prediction accuracy.

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 SDSC Paragon, this template generates categories such as (`q16m,wsmith`), (`q64l,wsmith`), and (`q16m,foster`).

Table 2
Characteristics recorded in workloads

| | 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 |

The column “Abbr” indicates abbreviations used in subsequent discussion.

## 2.2. Generating predictions

We now consider the question of how we generate runtime predictions. The input to this process is a set of templates $T$ and a workload $W$ for which runtime predictions are required. In addition to the characteristics described in the preceding section, a maximum history, type of data to store, 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 actual run time to the user-supplied estimate (as described in Section 2.3). The prediction type determines how a runtime prediction is made from the data in each category generated from a template. We consider four prediction types: a mean, denoted by `mean`, a linear regression (`lin`), an inverse regression (`inv`), or a logarithmic regression (`log`).

The output from this process is a set of run-time predictions and associated confidence intervals. 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 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 $c_i \notin C$, 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 can 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 regressions to compute coefficients $a$ and $b$ for the equations $R = aN+b$, $R = \frac{N}{a}+b$, and $R = a \log N+b$ for linear, inverse and logarithmic regressions, respectively. $N$ is the number of nodes requested by the jobs, and $R$ is the run time. The techniques used to compute confidence intervals for these predictors, are described in the appendix.

The use of maximum histories, denoted by `mh`, 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

One possible way to improve prediction accuracy is to incorporate run time estimates provided by users at the time of application 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. The ANL and CTC workloads include user-supplied maximum run times. 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 implicitly provide 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; 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. We take advantage of this information by predicting what fraction of the user run time estimate the application will execute.

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 |

### 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.

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 $i = 1 -> n$
    (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 $X_c$ with the lowest mean error, and add
        the associated template $t_c$ to $T$.
```

Our greedy algorithm can search over any set of characteristics.

### 2.4.2. Genetic algorithm

The second search algorithm that we consider uses genetic algorithm techniques [9] to achieve a more detailed exploration of the search space. 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.

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: the type of regression to use, whether to use absolute or relative run times, which characteristics to enable, the node range to use, and the amount of history to store in each category.

A fitness function is used to compute the fitness of each individual and therefore its chance to reproduce. 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 stochastic 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.

## 3. Experimental results

In the rest of this paper, we discuss experimental studies that we have performed to evaluate the effectiveness of our techniques and the significance of the refinements just noted.

### 3.1. Greedy search

Fig. 1 shows the results of our first set of greedy searches for template sets. The characteristics searched over are the ones listed in Table 2. Actual run times are used as data points and a curve is shown for each of the predictors. 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 25 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 1 to 20 percent with the ANL workload seeing the most benefit and the SDSC96 workload seeing the least.

Second, the graphs show that for the final template set, the mean is a better predictor than any of the regression. The

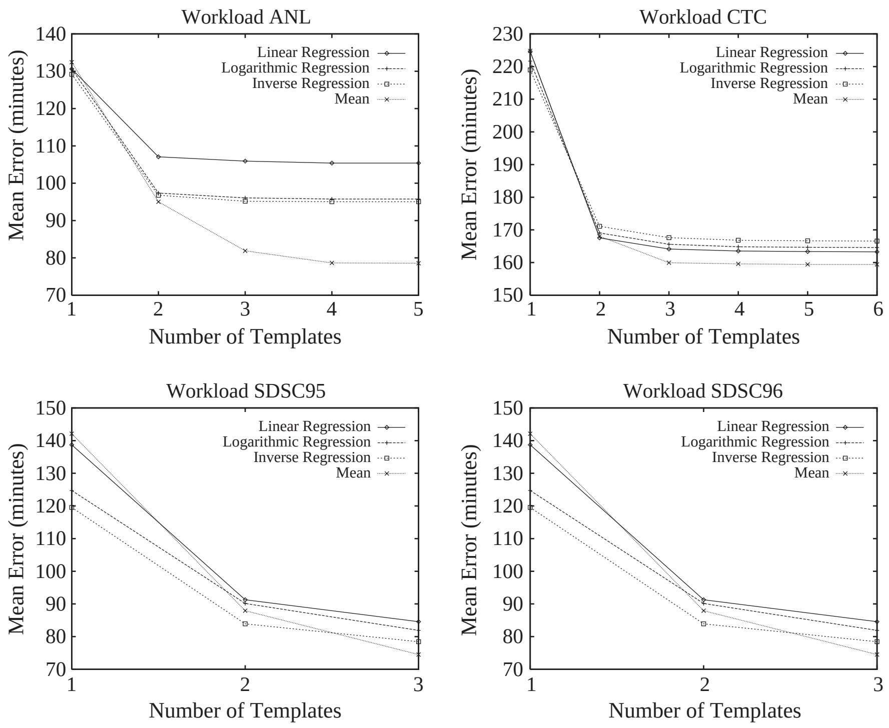

Fig. 1. The mean errors of the greedy searches using run times as data points.

final predictors obtained by using means are between 2 and 18 percent more accurate than those based on regressions. The impact of the choice of predictor on accuracy is greatest in the ANL workload and least in the CTC workload. If we search over the predictor as well as the other characteristics, the search results in a template set that is up to 11 percent better than any search using a particular predictor.

Fig. 2 shows searches performed over the ANL and CTC workloads when relative run times are used as data points. Similar to the data in Fig. 1, adding a second characteristic with a single characteristic results in the largest improvement in performance. Also, comparing the prediction errors in Fig. 2 with the prediction errors in Fig. 1 shows that using relative run times as data points results in a performance improvement of 19–48 percent.

Table 4 lists for each workload the accuracy of the best category templates found by the first set of greedy searches. 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 our subsequent searches perform 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 one application at a time on these parallel computers.

The templates selected for the SDSC workloads indicate that the user who submits an application is more important in determining application similarity than the queue to which an application is submitted. Furthermore, Fig. 1 shows that adding the third template results in performance improvements of only 2–12 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 is not contained in the trace data available to us.

We next performed a second series of greedy searches to identify the impact of using node information when defining categories. We used node ranges when defining categories as described in Section 2.1. The results of these searches in Table 5 show that using node information improves prediction performance by 1–9 percent for the best predictors, 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.

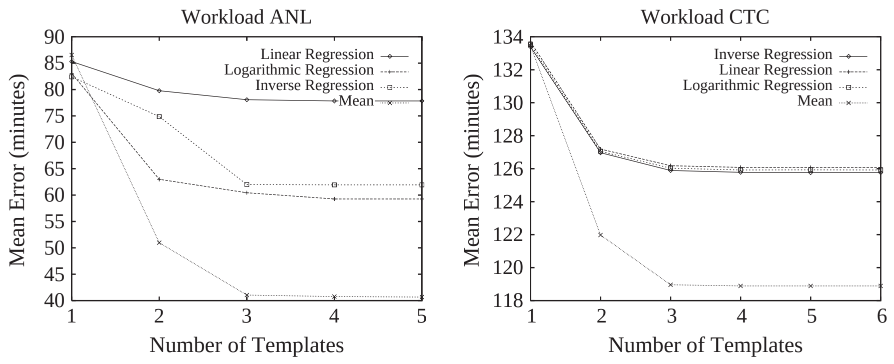

Fig. 2. The mean errors of the greedy searches using relative run times as data points.

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,u,a), (t,u,e,a)` | 40.68 | 41.77 |
| 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 |

Table 5
Best predictions found during second greedy search, using node information

| Workload | Predictor | Data point | Template set | Mean error (min) | Percentage of mean run time |
|---|---|---|---|---|---|
| ANL | Mean | Relative run time | (), (e), (u,a), (t,u, *n*=2), (t,e,a, *n* =16), (t,u,e,a, *n*=32) | 39.87 | 40.93 |
| CTC | Mean | Relative run time | (), (u), (e,*n*=1), (t,u,*n*=1) (c,u,e, *n*=8), (t,c,u,ni, *n*=4) (t,c,u,e,ni, *n*=256) | 117.97 | 64.75 |
| 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 |

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.

Our third and final series of searches identify the impact of setting a maximum amount of history to use when making predictions from categories. The results of these searches are shown in Table 6. Comparing this data with the data in Table 5 shows that using a maximum amount of history improves prediction performance by 14–34 percent for the best predictors. The least improvement occurs for the ANL workload, the most for the SDSC96 workload. Other facts to note about the results in the table are that a maximum history is used in the majority of the templates and that when a maximum history is used, it is relatively small. The latter fact indicates that temporal locality is relevant in the workloads.

### 3.2. Genetic algorithm search

We now investigate the effectiveness of using a genetic algorithm to search for the best template sets by performing the same three series of searches. The results are shown in Tables 7–9 along with the results of the corresponding greedy searches for comparison.

As shown in the tables, the best templates found during the genetic algorithm search provide mean errors that are 10 percent better to 1 percent worse than the best templates found during the greedy search. For the majority of the experiments, the genetic search outperforms the greedy search.

The best template sets identified by the genetic search procedure are listed in Table 10. This data shows that similar to the greedy searches, the mean is the single best predictor to use and using relative run times as data points, when available, provides the best performance.

Table 6
Best predictions found during third greedy search

| Workload | Predictor | Data point | Template set | Mean error (min) | Percentage of mean run time |
|---|---|---|---|---|---|
| ANL | Mean | Relative run time | (),(e),(u,mh=4), (t,u,mh=16), (e,a,n=64,mh=4), (t,e,a,n=1,mh=8), (t,u,e,a,n=16,mh=8) | 34.28 | 35.20 |
| SDSC95 | Mean | Run time | (),(u),(q,mh=4), (u,n=8,mh=8), (q,u,n=1,mh=32) | 48.33 | 45.16 |
| SDSC96 | Mean | Run time | (),(u),(q,mh=4), (u,n=4,mh=4) (q,u,n=1,mh=16) | 50.14 | 30.12 |

Table 7
Performance of the best templates found during first genetic algorithm search

| Workload | Genetic algorithm | | Greedy | |
|---|---|---|---|---|
| | Mean error (min) | Percentage of mean run time | Mean error (min) | Percentage of mean run time |
| ANL | 39.32 | 40.37 | 40.68 | 41.77 |
| CTC | 107.02 | 58.74 | 118.89 | 65.25 |
| SDSC95 | 65.27 | 60.57 | 75.56 | 70.12 |
| SDSC96 | 80.37 | 48.28 | 82.40 | 49.50 |

Table 8
Performance of the best templates found during second genetic algorithm search

| Workload | Genetic algorithm | | Greedy | |
|---|---|---|---|---|
| | Mean error (min) | Percentage of mean run time | Mean error (min) | Percentage of mean run time |
| ANL | 38.79 | 39.83 | 39.87 | 40.93 |
| CTC | 106.25 | 58.32 | 118.05 | 64.80 |
| SDSC95 | 60.03 | 55.71 | 67.63 | 62.76 |
| SDSC96 | 74.75 | 44.90 | 76.20 | 45.77 |

Table 9
Performance of the best templates found during third genetic algorithm search

| Workload | Genetic algorithm | | Greedy | |
|---|---|---|---|---|
| | Mean error (min) | Percentage of mean run time | Mean error (min) | Percentage of mean run time |
| ANL | 34.52 | 35.44 | 34.28 | 35.20 |
| CTC | 98.28 | 53.95 | No data | No data |
| SDSC95 | 43.20 | 40.09 | 48.33 | 45.16 |
| SDSC96 | 47.47 | 28.51 | 50.14 | 30.12 |

Table 10
The best templates found during genetic algorithm search

| Workload | Predictor | Data point | Template set |
|---|---|---|---|
| ANL | Mean | Relative run time | (u,e,n=128,mh=16384), (u,e,n=16,mh=4), (t,e,n=16,mh=128), (t,u,n=4,mh=inf), (t,u,a,n=4,mh=4), (t,u,a,n=64,mh=4), (t,u,e,a,n=64,mh=32) |
| CTC | Mean | Relative run time | (u,n=64,mh=8), (c,s,n=32,mh=128), (c,s,n=32,mh=256), (c,u,ni,n=256,mh=128), (t,u,s,n=1,mh=16), (t,c,u,ni,n=256,mh=32), (t,c,u,ni,n=4,mh=16384), (t,c,u,s,ni,n=1,mh=4) |
| SDSC95 | Mean | Run time | (q,u,n=4,mh=4), (q,u,n=4,mh=64), (q,u,n=4,mh=8) (q,u,n=16,mh=1024), (q,u,n=2,mh=4), |
| SDSC96 | Mean | Run time | (q,u,n=4,mh=1024), (q,u,n=4,mh=2048), (q,u,n=4,mh=4096), (q,u,n=4,mh=65536), (q,u,n=4,mh=8) |

Another observation is that node information and a maximum history are used throughout the best templates found during the genetic search. This confirms the observation made during the greedy search that using this information when defining templates results in improved prediction performance.

Fig. 3 shows the progress of the two different search algorithms for a search of template sets for the ANL workload that use the number of nodes requested, limit the maximum history, either actual or relative run times, and use any of the predictors. The graph shows that while both searches result in template sets that have nearly the same accuracy, the genetic algorithm search does so much more quickly. This fact is important because a simulation that takes minutes or hours is performed to evaluate each template set.

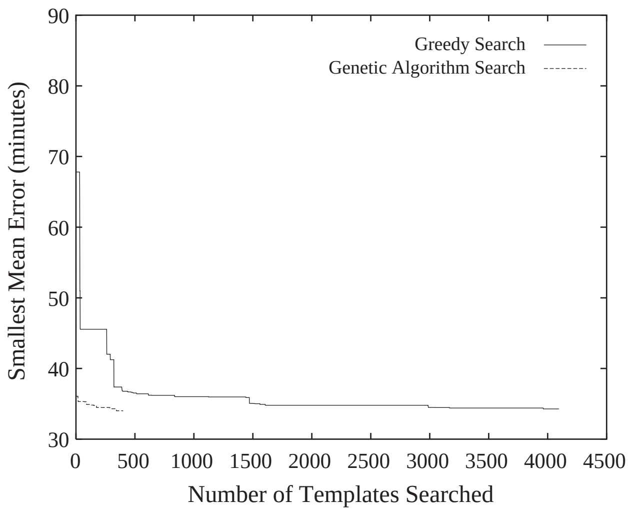

Fig. 3. Search efficiency for workload ANL.

## 4. Related work

Gibbons [7,8] also uses historical information to predict the run times of parallel applications. He 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 `rtime` (how long an application has executed). Rather than having equal-sized ranges specified by a parameter, as we do, Gibbons defines the fixed set of exponential ranges 1, 2–3, 4–7, 8–15, and so on. Further, 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 11 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 between 4 and 59 percent better than Gibbons's technique and our genetic algorithm search finds template sets that have between 21 and 61 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 12 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 searches.

Table 11
Comparison of our prediction technique with that of Gibbons

| Workload | Gibbons's mean error (min) | Our mean error | |
|---|---|---|---|
| | | Greedy search (min) | Genetic algorithm (min) |
| ANL | 75.26 | 34.28 | 34.52 |
| CTC | 124.06 | 117.97 | 98.28 |
| SDSC95 | 74.05 | 48.33 | 43.20 |
| SDSC96 | 122.55 | 50.14 | 47.47 |

Table 12
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 (min) | Our mean error | |
|---|---|---|---|
| | | Greedy search (min) | Genetic algorithm (min) |
| ANL | 49.47 | 34.28 | 34.52 |
| CTC | 107.41 | 117.97 | 98.28 |

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.

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. The second technique uses the conditional average lifetime.

The performance of both of these techniques are shown in Table 13. We have reimplemented Downey's technique as described in [3] 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 45–64 percent better than 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 running times while we concentrate on accurately predicting jobs with a running time of 0.

Table 13
Comparison of our prediction technique with that of Downey

| Workload | Downey's mean error | | Our mean error | |
|---|---|---|---|---|
| | Conditional median lifetime (min) | Conditional average lifetime (min) | Greedy search (min) | Genetic algorithm (min) |
| ANL | 97.01 | 106.80 | 34.28 | 34.52 |
| CTC | 179.46 | 201.34 | 117.97 | 98.28 |
| SDSC95 | 82.44 | 171.00 | 48.33 | 43.20 |
| SDSC96 | 102.04 | 168.24 | 50.14 | 47.47 |

## 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 29–54 percent of mean run times in four supercomputer center workloads. The greedy search finds templates that result in prediction errors of 30–65 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 21–61 percent lower error than those obtained by Gibbons and 45–64 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 19–48 percent improvement in performance for the Argonne and Cornell workloads. This suggest a simple way to greatly improve prediction performance: ask users to provide their own predictions and use this information when calculating predictions. We used both means and three types of regressions to produce run-time estimates from similar past applications and found that means are the single most accurate predictor but using a combination of predictors improves performance. For the best templates found in the greedy search, using the mean for predictions resulted in between 2 and 18 percent smaller errors, and using all predictors resulted in no improvement to 11 percent improvement.

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 important characteristics to know about jobs. Predicting based on the number of nodes only improves performance by 2–9 percent. We also find that there is temporal locality, and hence specifying a maximum history improves prediction performance by 14–34 percent.

In other work, we apply our run-time prediction techniques to the problems of selecting and co-allocating resources in metacomputing systems [1,5,6]. For example, using run-time predictions to predict queue wait times will allow users to select resources based on predictions of when their applications will execute [13]. Further, run-time predictions may improve scheduling efficiency when reserving resources for applications on systems with queuing schedulers [14].

## Acknowledgements

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 performed at Argonne National Laboratory and Northwestern University and 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.

## References

[1] C. Catlett, L. Smarr, Metacomputing, Commun. ACM 35 (6) (1992) 44–52.
[2] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metasystems, Lecture Notes on Computer Science, Springer, Berlin, 1998.
[3] A. Downey, Predicting queue times on space-sharing parallel computers, in: International Parallel Processing Symposium, 1997.
[4] D. Feitelson, B. Nitzberg, Job characteristics of a production parallel, scientific workload on the NASA ames iPSC/860, Lecture Notes on Computer Science, vol. 949, Springer, Berlin, 1995, pp. 337–360.
[5] Ian Foster, Carl Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kauffmann, 1999.
[6] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, Internat. J. Supercomput. Appl. 11 (2) (1997) 115–128.

[7] R. Gibbons, A historical application profiler for use by parallel schedulers, Lecture Notes on Comput. Science, vol. 1297, Springer, Berlin, 1997, pp. 58–75.
[8] R. Gibbons, A historical profiler for use by parallel schedulers, Master's Thesis, University of Toronto, 1997.
[9] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[10] W. Smith, Resource management in metacomputing environments, Ph.D. Thesis, Northwestern University, December 1999.
[11] W. Smith, I. Foster, V. Taylor, Scheduling with advanced reservations, in: Proceedings of the 2000 International Parallel and Distributed Processing Symposium, May 2000.
[12] W. Smith, V. Taylor, I. Foster, Using run-time predictions to estimate queue wait times and improve scheduler performance, in: Proceedings of the IPPS/SPDP'99 Workshop on Job Scheduling Strategies for Parallel Processing,1999.
[13] Warren Smith, Valerie Taylor, Ian Foster, Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance, in: Dror Feitelson, Larry Rudolph (Eds.), Lecture Notes on Computer Science, vol. 1659, Springer-Verlag, 1999, pp. 202–219.
[14] Warren Smith, Ian Foster, Valerie Taylor, Scheduling with Advanced Reservations, in: Proceedings of the 2000 International parallel and Distributed Processing Symposium, May 2000.

**Warren Smith** is currently a senior computer scientist working for Computer Sciences Corporation in the NASA Advanced Supercomputing Division of NASA Ames Research Center. His research interests lie in the area of distributed systems including performance prediction, scheduling, information systems, and event management. He received B.S. and M.S. degrees from the Johns Hopkins University and M.S. and Ph.D. degrees from Northwestern University.

**Ian Foster** holds a joint appointment as the Associate Division Director of the Mathematics and Computer Science Division at Argonne National Laboratory and Arthur Holly Compton Professor of Computer Science at the University of Chicago. His research is focused on tools and techniques that allow people to use high-performance computing technologies to do qualitatively new things. This involves investigations of parallel and distributed languages, algorithms, and communication; and also focused work on applications. Ian Foster is particularly interested in using high-performance networking to incorporate remote compute and information resources into local computational environments.

**Valerie E. Taylor** earned her Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley, in 1991. From 1991-2002, she was a member of the faculty of Northwestern University. Valerie Taylor has since joined the faculty of Texas A&M University as Head of the Texas A&M Engineering Department of Computer Science and holder of the Stewart & Stevenson Professorship II. Her research interests are in the areas of computer architecture and high performance computing, with particular emphasis on mesh partitioning for distributed systems and the performance of parallel and distributed applications. She has authored or co-authored over 80 publications in these areas. Valerie Taylor is a senior member of IEEE and a member of ACM.