Predicting queue times on space-sharing parallel computers

Allen B. Downey *

Abstract

We present statistical techniques for predicting the queue times experienced by jobs submitted to a space-sharing parallel machine with first-come-first-served (FCFS) scheduling. We apply these techniques to trace data from the Intel Paragon at the San Diego Supercomputer Center and the IBM SP2 at the Cornell Theory Center. We show that it is possible to predict queue times with accuracy that is acceptable for several intended applications. The coefficient of correlation between our predicted queue times and the actual queue times from simulated schedules is between 0.65 *and* 0.72.

1 Introduction

On space-sharing parallel computers, it is useful to be able to predict how long a submitted job will be queued before processors are allocated to it. Some of the applications of these predictions are:

Load metrics: They provide a measure of load that is more concrete than abstractions such as load average, allowing users to make decisions about what jobs to run, where to run them or what size problems they can solve in an allotted time.

Internal resource selection: They allow malleable parallel jobs (jobs that do not require a specific number of processors, but can run on a range of cluster sizes) to choose a cluster size that is appropriate for the current state of the machine. This type of allocation is also called *adaptive partitioning*.

External resource selection: They allow distributed jobs to choose among various computing resources on a network, based on the quality of service they expect to receive at each host. As part of the DOCT project [13] we are planning to implement the techniques proposed here to support resource selection in a distributed, heterogeneous environment.

For different applications, we will make our predictions at different times: for external resource selection, we need to predict the entire queue time from arrival to beginning of execution; for internal resource selection we will consider only the time from arrival at the head of the queue until beginning of execution, which is sometime called *wait time*.

The focus of this paper is internal resource selection, and thus we will be making predictions when jobs arrive at the head of the queue. In future work we will extend these techniques to include the entire time a job spends in queue, and apply those predictions to external resource selection.

1.1 Internal resource selection

Large parallel computers are usually shared by many users. Two mechanisms that support this sharing are

Space-sharing: The machine may be partitioned into sets of processors (clusters). Each cluster is allocated to a single job that is allowed to run to completion (RTC).

Time-sharing: More than one job may be allocated to a cluster, in which case each job runs for some quantum of time before being preempted to allow other jobs to run.

The advantages of space-sharing are a dedicated system image (a cluster behaves like a small dedicated system), low overheads (preemption is expensive) and high parallel efficiencies (since jobs tend to run on fewer processors). One disadvantage is that RTC scheduling sometimes allows short jobs to sit in queue waiting for long jobs, resulting in poor average turnaround time. The systems we are studying, the Intel Paragon at the San Diego Supercomputer Center and the IBM SP2 at the Cornell Theory Center, are configured as space-sharing systems using RTC scheduling.

Many programs running on these systems are *malleable*, meaning that they can run on a range of cluster sizes. At present few if any of these jobs can change cluster sizes

*Computer Science Division, University of California, Berkeley, CA 94720 and San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186. Supported by NSF grant ASC-89-02825 and by Advanced Research Projects Agency/ITO, Distributed Object Computation Testbed, ARPA order No. D570, Issued by ESC/ENS under contract #F19628-96-C-0020. email: downey@sdsc.edu, http://www.sdsc.edu/~downey dynamically; once they begin execution, they do not yield or allocate processors. In this context, internal resource selection means choosing a cluster size for each job before it begins execution.

When a job arrives at the head of the queue, it may allocate some or all of the processors that are currently free, or it may wait for additional processors to become available. If it chooses to wait, we assume for now that the other jobs in queue must also wait. In other words, we assume first-come-first-served (FCFS) queueing. In future work, we will relax this assumption and address more general queueing policies.

In summary, we use the following model of computation:

- There is no timesharing. Jobs allocate processors exclusively and are not preempted until they complete (or exceed their allocated time).
- At least some jobs in the system are malleable, but once they begin execution, they cannot change cluster sizes dynamically.
- If there are not enough free processors for a job, it waits in a first-come-first-serve (FCFS) queue.
- Jobs in different partitions are independent. We do not consider communication interference.

Our approach to internal resource selection is to allow each job, when it arrives at the head of the queue, to choose the cluster size that minimizes its turnaround time, which is the sum of its queue time and run time. To make this decision, the job must know its run time on a range of cluster sizes, and predict the queue time until each cluster size is available.

In [4] we develop a speedup model that predicts run times; in this paper we present techniques for predicting queue times. We make these predictions in two steps: first, we develop a *conditional lifetime model* that predicts the remaining lifetime of a job based on its current age. Then we aggregate these job-by-job predictions into a queue time prediction.

1.2 Overview

Section 2 describes related work. Section 3 presents the distribution of lifetimes of jobs on the Intel Paragon at SDSC, and proposes the conditional lifetime model (CLM) we use to predict the remaining lifetimes of jobs. In Section 4 we use this CLM to predict the wait time for a given cluster size. Section 5 describes a trace-driven simulator we used to evaluate these techniques and presents our results.

In Section 6 we apply the proposed techniques in another environment (the IBM SP2 at the Cornell Theory Center). Section 7 summarizes our findings and describes our plan for continuing work.

2 Related work

To predict a job's duration it is useful to know the distribution of lifetimes for all jobs. For sequential jobs on UNIX workstations, two studies have reported distributions of lifetimes with the functional form

$$cdf_L(t) = Pr\{L < t\} = 1 - (t/t_{min})^k \qquad (1)$$

where k is a parameter that varies from workload to workload [12] [8]. When k is approximately 1, as it often is, the median remaining lifetime of a job is equal to its current age. This property is known as the *past-repeats heuristic.* In the next section, we will show that the distribution of lifetimes for parallel scientific applications does not fit this model; thus, the past-repeats heuristic does not apply in these environments.

Several previous studies have reported lifetime distributions for parallel scientific applications, but none discuss the shape of the lifetime distribution or use it to develop a workload model. Hotovy *et al.* [10] [11] describe the workload on the IBM SP2 at the Cornell Theory Center. Steube and Moore [14] and Wan *et al.* [15] describe the workload and scheduling policy on the Intel Paragon at the San Diego Supercomputer Center; in [14] the distribution of lifetimes for batch jobs appears to fit the uniform-log model proposed here (for jobs less than six hours in duration). Feitelson and Nitzberg [6] describe the workload on the Intel iPSC/860 at NASA Ames. Finally, Windisch *et al.* [16] compare the workloads from SDSC and NASA Ames.

Devarakonda and Iyer [3] use information about past executions of a program to predict the lifetimes of UNIX processes, as well as their file I/O and memory use.

Atallah *et al.* [1] propose the idea of choosing a cluster size that minimizes the turnaround time (queue time plus run time) of each job. Since their target system is a timesharing network of workstations, they consider the problem of contention with other jobs in the system, but they do not have the problem of predicting the time until a cluster becomes available. Like us, they raise the question of how local optimization by greedy users affects overall system performance. We address this question in [5]; we allow each job to choose the cluster size that minimizes its expected turnaround time, and find that this application-centric strategy leads to global performance better than that of many proposed system-centric strategies.

3 Conditional lifetime model (CLM)

During 1995, 24906 jobs were submitted to the batch partition of the 400-node Intel Paragon at SDSC. Figure 1 shows the distribution of lifetimes for these jobs.

Except for the longest and shortest jobs, this distribution is nearly straight (on a logarithmic x-axis); in other words,

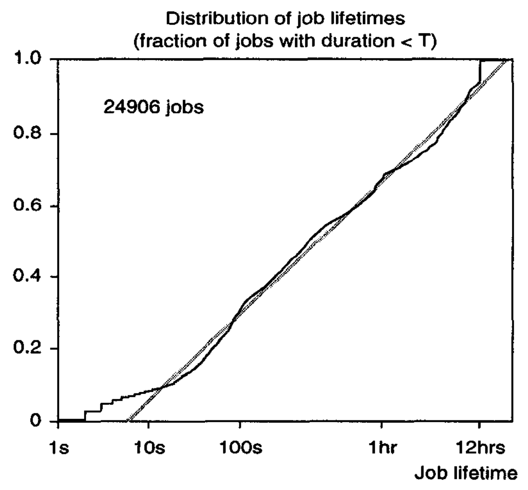


Figure 1. The distribution of lifetimes for 24906 batch jobs submitted to the Intel Paragon at SDSC. The gray line shows the least-squares fit for the observed data (intercept β_0 = -0.18, slope β_1 = 0.10).

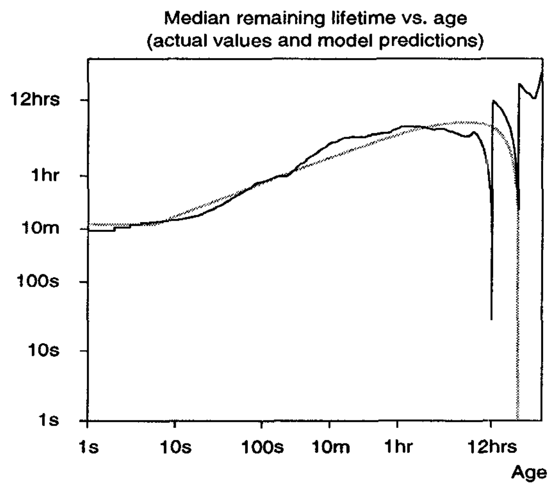


Figure 2. The median remaining lifetime of a job as a function of its current age. The gray line shows the values predicted by our model, with the estimated parameters from Figure 1.

the logs of the lifetimes are distributed uniformly. Although we do not know any theoretical reason why lifetimes should have this *uniform-log distribution*, we will take advantage of this observation by fitting a straight-line approximation to the observed distribution. Using this approximation (instead of the observed values) simplifies the calculation of the conditional distribution of lifetimes (Equation 3).

We calculated a straight-line approximation by least-squares regression (omitting the longest 10% and the shortest 10% of jobs). The R^2 value of the fit is over 0.99, indicating that the linear approximation is very good. Thus our model of this distribution of lifetimes is

$$cdf_L = Pr\{L < t\} = \beta_0 + \beta_1 \ln t \tag{2}$$

where cdf_L is the cumulative distibution function of L, a random variable representing the lifetime of a job, and β_0 and β_1 are the parameters of the model (intercept and slope). Also, t is bounded by $t_{min} = e^{-\beta_0/\beta_1}$ and $t_{max} = e^{(1-\beta_0)/\beta_1}$. For the estimated parameters from Figure 1, $t_{min} = 5.8$ seconds and $t_{max} = 24.2$ hours.

For this model we can calculate the distribution of lifetimes conditioned on the current age of a job:

$$\begin{aligned} 1 - cdf_{L|a} &= Pr\{L > t | L > a\} \\ &= \frac{1 - cdf_L(t)}{1 - cdf_L(a)} \\ &= \frac{1 - \beta_0 - \beta_1 \ln t}{1 - \beta_0 - \beta_1 \ln a} \end{aligned} \tag{3}$$

To find the median lifetime of a job given its current age, we set the conditional cdf to 0.5 and solve for t:

$$\begin{aligned} cdf_{L|a} &= \frac{1 - \beta_0 - \beta_1 \ln t}{1 - \beta_0 - \beta_1 \ln a} = 0.5 \\ \ln t &= .5 \frac{1 - \beta_0}{\beta_1} + .5 \ln a \end{aligned} \tag{4}$$

Substituting $t_{max} = e^{(1.0-\beta_0)/\beta_1}$, we express the median lifetime as $\sqrt{t_{max} \cdot a}$. Thus, this distribution does not obey the past-repeats heuristic; rather, the median remaining lifetime increases slowly with age, until $t > \frac{1}{4} t_{max}$, after which the median remaining lifetime drops quickly.

Figure 2 shows the median remaining lifetime as calculated according to the model and according to the observed data. The proposed model fits the data well for ages less than 6 – 12 hours. For longer lifetimes, the distribution of lifetimes diverges from the model. The remaining lifetimes of old jobs do not decline, as predicted, but appear to increase. It is difficult to generalize this behavior, because there are few jobs with lifetimes greater than 12 hours[1] and their distribution does not fit our model.

The average remaining lifetime of a job, $E[L|a] = (t_{max} - a)/(\ln t_{max} - \ln a)$, behaves similarly; it increases slowly with age and then drops sharply as the age approaches the limit. Since our model is not accurate for jobs with lifetimes greater than 12 hours, and since the distribution of these jobs is difficult to characterize, we will need to be careful about how we apply our model to old jobs.

[1]For most jobs the time limit was 12 hours, but a few jobs (less than 1%) were allowed to run 24 hours or longer by special permission. The time limits at 12 and 24 hours account for the spikes in the curve.

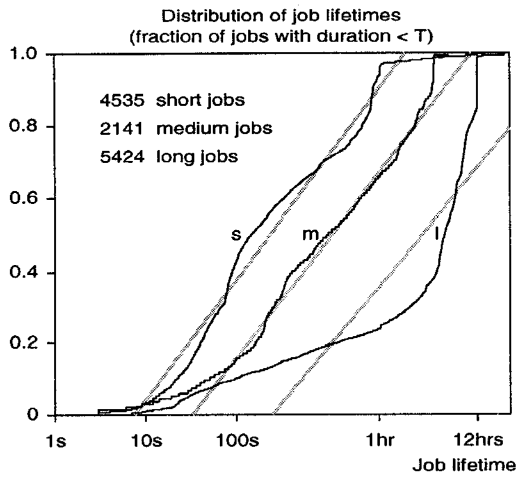


Figure 3. The distribution of lifetimes for parallel batch jobs submitted to the Intel Paragon at SDSC, broken down by queue. The gray lines show the least-squares fit for each queue.

3.1 Using additional information about jobs

If jobs can be divided into classes on the basis of *a priori* information, and the distribution of lifetimes differs between classes, then it is possible to improve the accuracy of the predictions by estimating and using different workload parameters for each class.

Depending on the environment, jobs might be partitioned according to executable name, user name, or declared resource requirements. Many queueing systems ask users to provide estimates of the resource requirements of their jobs: *e.g.*, number of processors, memory requirements, and estimated run times.

On the Intel Paragon at SDSC, users declare the resource requirements of their jobs implicitly by their selection of one of the NQS queues. Most jobs are submitted to queues that specify the maximum cluster size of the job (powers of two) and the length of time it will run (short, medium, or long).

Not surprisingly, we found that the distributions for the short, medium and long queues are significantly different (Figure 3). The differences among these distributions indicate that users have some information about the expected run times of their jobs; on the other hand, this information is far from perfect. There is considerable overlap between the different queues — many jobs are submitted to what turns out to be the wrong queue. For example, 30% of the jobs from the short queue run longer than the median of the medium queue, and 17% of the jobs from the medium queue are shorter than the median of the short queue.

Thus it is not obvious how to use queue information to predict the remaining lifetime of a job. The technique we are proposing, using a different conditional distribution for each queue, seems like an effective way to use information provided by the user without suffering greatly if that information turns out to be wrong.

Using the same technique as in the previous section (discarding the longest and shortest 10% of each group), we fit a line to the lifetime distribution for each queue. Table 1 shows the resulting coefficients. We treat sequential jobs as a distinct class because their distribution of lifetimes is significantly different from that of the parallel jobs.

For all but the long queue, the goodness of fit metric is quite good; for the long queue, it is only 0.74, which is not surprising, since the distribution is obviously not a straight line. The distribution for long jobs is bimodal: roughly 40% run between 10 seconds and 3.5 hours; the remaining 60% run between 3.5 and 12 hours. In Section 5.4 we extend our model to address this poor fit, but in the meantime we will use the estimated parameters as is.

Table 1

	β_0	β_1	R^2
sequential	-.65	.21	.99
short	-.30	.15	.97
medium	-.50	.14	.99
long	-.73	.13	.74

4 Calculating the distribution of queue times

Given the state of a system — the number of jobs currently running, their ages and their cluster sizes — we can use the CLM to estimate the wait time until a given cluster size, n, is available.

We describe the state of the machine at the time of an arrival as follows: there are p jobs running, with ages a_i and cluster sizes n_i (in other words, the ith job has been running on n_i processors for a_i seconds). We would like to predict $Q(n')$, the time until n' additional processors become available, where $n' = n - n_{free}$ and n_{free} is the number of free processors.

In the next two sections we will present ways to estimate the median and mean of $Q(n')$. Then we will use these values to predict wait times during a trace-driven simulation.

4.1 Predictor A: median

We can calculate the median of $Q(n')$ exactly by enumerating all possible outcomes (which jobs complete and which are still running), and calculating the probability that the request will be satisfied before time t. Then we can find the median queue time by setting this probability to 0.5 and solving for t. This approach is not feasible when there are many jobs in the system, but it leads to an approximation that is fast to compute and that turns out to be sufficiently accurate for our intended purposes.

We represent each outcome by a bit vector, b, where for each bit, $b_i = 0$ indicates that the ith job is still running, and $b_i = 1$ indicates that the ith job has completed before time t. Since we assume independence between jobs in the system, the probability of a given outcome is the product of the probabilities of each event (the completion or non-completion of a job), and the probability of these events come from the CLM.

$$Pr\{b\} = \prod_{i|b_i=0} cdf_{L|a_i}(t) \cdot \prod_{i|b_i=1} (1 - cdf_{L|a_i}(t)) \quad (5)$$

For a given outcome, the number of free processors is the sum of the processors freed by each job that completes:

$$F(b) = \sum_i b_i \cdot n_i \quad (6)$$

Thus at time t, the probability that the number of free processors is at least the requested cluster size is the sum of the probabilities of all the outcomes that satisfy the request:

$$Pr\{F \geq n'\} = \sum_{b|F(b)\geq n'} Pr\{b\} \quad (7)$$

Finally, we find the median value of $Q(n')$ by setting $Pr\{F > n'\} = 0.5$ and solving for t.

Of course, the number of possible outcomes (and thus the time for this calculation) increases exponentially with p. Thus it is not a feasible approach when there are many running jobs. But when n' is small, it is often the case that there are several jobs running in the system that will single-handedly satisfy the request when they complete. In this case, the probability that the request will be satisfied by time t is dominated by the probability that any one of these benefactors will complete.

In other words, the chance that the queue time for n' processors will exceed time t is approximately equal to the probability that none of the benefactors will complete before t:

$$Pr\{F < n'\} \approx \prod_{i|n_i\geq n'} 1 - cdf_{L|a_i}(t) \quad (8)$$

This calculation is only approximately correct, since it ignores the possibility that several small jobs might complete and satisfy the request. Thus, we expect this predictor to be inaccurate when there are many small jobs running in the system, few of which can single-handedly satisfy the request. The next section presents an alternative predictor that we expect to be more accurate in this case.

4.2 Predictor B: mean

When a job is running, we know that at some time in the future it will complete and free all of its processors. Given the age of the job, we can use the CLM to calculate the probability that it will have completed before time t.

We approximate this behavior by a model in which processors are a continuous (rather than discrete) resource, which jobs release gradually as they execute. In this case, we imagine that the CLM indicates what *fraction* of a job's processors will be available at time t.

For example, a job that has been running on 10 processors for 6 minutes might have a 30% chance of completing in the next 2 minutes, releasing all ten of its processors. As an approximation of this behavior, we predict that the job will release 30% of its processors within the next two minutes.

Thus we predict that the number of free processors at time t will be the sum of the processors released by each job:

$$F = \sum_i n_i \cdot cdf_{L|a_i}(t) \quad (9)$$

Then to estimate the expected (mean) queue time we set $F = n'$ and solve for t.

5 Trace-driven simulator

To evaluate the accuracy of these predictors, we ran trace-driven simulations of workloads from the Intel Paragon at the San Diego Supercomputer Center. As each job arrived at the head of the queue, we calculated both predictors based on the observed state of the system, and compared the predictions with the queue times that followed.

We obtained a trace of 24906 jobs submitted to the batch partition of the Paragon between January 1, 1995 and December 31, 1995. Using the arrival times, cluster sizes and durations of these jobs, we simulated their execution on a parallel computer with 368 nodes (the size of the batch partition at SDSC). This data and our simulator are available from http://sdsc.edu/~ downey/predict.

Table 2 shows the average and median lifetimes and cluster sizes for these jobs. CV is the coefficient of variation — the ratio of the standard deviation to the mean. The 25th and 75th percentiles are also shown.

Table 2

	avg	CV	25%	median	75%
lifetime (s)	9020	1.74	77	720	14500
cluster size	25	1.47	1	16	32

The jobs in these traces are not malleable (at least not from the system's point of view). Users choose the cluster size for each job; the system cannot allocate fewer, and does not allocate more.

Although the arrival times of the jobs are taken from the traces, the schedule used by the simulator differs from the actual schedule that was executed on the Paragon:

- The Paragon at SDSC assigns different priorities to each queue and schedules higher priority jobs first. Our simulator uses strict FCFS scheduling.
- The real scheduler allocates processors using a modified 2-D buddy system based on power-of-two partition sizes; most, but not all, jobs are allocated a contiguous set of nodes. The simulator allocates jobs without regard to contiguity.
- Most nodes have 16MB of memory, but 128 "fat" nodes have 32MB. In reality, some jobs can only run on the fat nodes, but the simulator ignores this distinction.
- During the year, the size of the batch partition changed several times; in the simulator we held the number of processors constant.

Thus, in evaluating our predictors, the "actual" queue times are from the schedule generated by the simulator, not from the trace data.

In our simulations, many jobs arrive at the head of the queue and find that there are enough free processors for them to begin execution immediately. Since these jobs spend no time at the head of the queue, we do not make predictions on their behalf. In a system with malleable jobs, though, cluster sizes are not determined *a priori*; they are chosen according to the state of the system (number of free processors, expected queue times, *etc.*). Each time a job arrives at the head of the queue, we predict the queue time for a range of cluster sizes and choose the one that best satifies the requirements of the job, perhaps by minimizing its expected turnaround time.

5.1 Results: Predictor A

When a job reaches the head of the queue and there are not enough free processors to satisfy its request, we calculate n', the number of additional processors required. We then consider the number of jobs in the system with cluster sizes greater than n'; that is, the jobs that will single-handedly satisfy the request if they complete.

In our simulations, the number of processors required tended to be small (50% below 16), and thus the number of potential benefactors was often large (75% of the time there are 3 or more). Under these circumstances, we expect Predictor A to perform well. Table 3 shows the average and median values of n' and the number of benefactors.

Table 3

	average	25%	median	75%
n'	30.6	7	16	45
benefactors	8.1	3	6	11

Of the 8545 times that a job waited at the head of the queue, 393 times it found no jobs running that could satisfy the request. Under these circumstances, Predictor A cannot make a prediction.

The remaining 8152 predictions are shown in Figure 4a, which is a scatter plot of predicted times on the x-axis and actual queue times on the y-axis. There is a clear correlation between the predicted and actual values; the coefficient of correlation (CC) between them is 0.63.

The solid line in the figure shows the mean of the actual queue times in each column; the broken line shows the median. For predicted values between 5 seconds and 3000 seconds, the mean and median values closely follow the 45-degree line, indicating that the predictions are accurate, at least on average. For predicted values greater than 3000 seconds, the predictions tend to be too high. This bias reflects the inherent conservative nature of the approximation: the predictor ignores the possibility of multiple jobs completing in order to fulfill a request. When there are few jobs that can satisfy a request single-handedly, the predictions tend to be too high.

Throughout the range, the median and mean are similar, indicating that the distribution of actual queue times in each column is roughly symmetric. In other words, for a given prediction, the actual queue time is equally likely to be higher or lower by a given amount.

5.2 Results: Predictor B

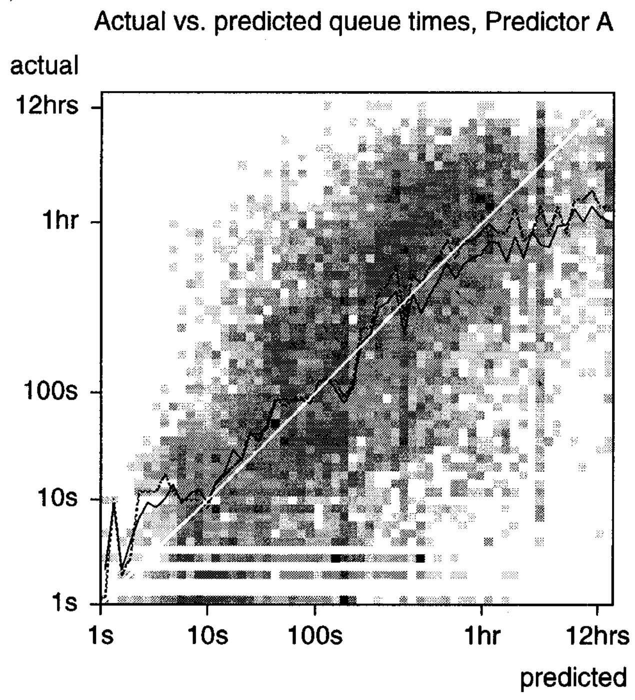
When there are many small jobs in the system we expect Predictor B to provide more accurate estimated queue times. Figure 4b shows a scatter plot of these predictions and the actual values.

In general, the correlation here is about as good as for Predictor A ($CC = 0.61$). For short queue times, though, the predicted values tend to be too low. This reflects the optimism of the approximation: instead of having to wait for a job to complete and release all its processors, the approximation is based on the assumption that jobs are constantly giving up some of their processors. Thus, for small values of n' this predictor tends to guess too low.

5.3 Combining the predictors

Since the two predictors perform well under different circumstances, it is natural to combine them by using each when it is most accurate. Comparing the values of CC for various ranges of n', we find that Predictor A does best for small values of n' while Predictor B does best for larger values. Using Predictor A when n' is less than 32 and Predictor B when it is greater, we get a combined predictor that is more accurate than either predictor alone ($CC = 0.65$).

a)



b)

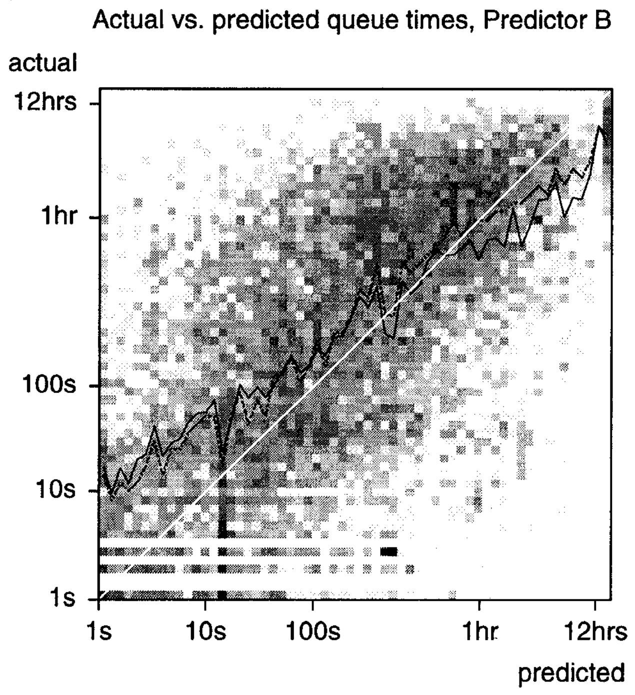


Figure 4. Scatterplot of actual queue times and predictions calculated by (a) Predictor A and (b) Predictor B. The white line shows the identity function; i.e. a perfect predictor. The solid line shows the average of the actual queue times in each column. The broken line shows the median.

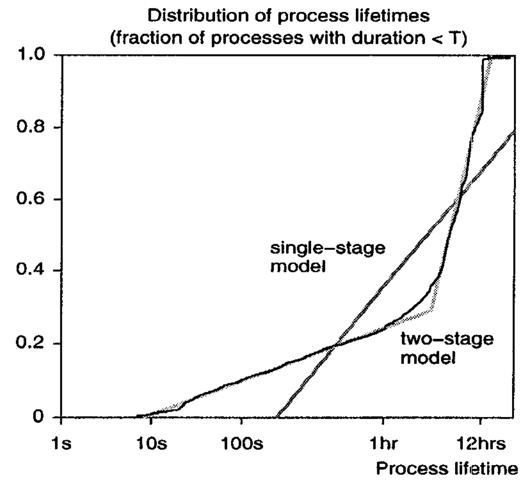


Figure 5. The distribution of jobs submitted to the long queue on the Intel Paragon at SDSC. The single-stage model is the same as in Figure 3. The two-stage model was chosen by hand to fit the observed data.

5.4 An improved model for long jobs

In Section 3.1 we observed that the uniform-log distribution model does not fit the distribution of jobs from the long queue well. In this section we develop an alternate model that fits this distribution well, and evaluate its effect on the accuracy of our predictors.

The uniform-log model can be extended to a multistage model that fits any piecewise linear distribution. For example, the observed distribution appears to contain two classes of job: short (between 10 seconds and 3 hours) and long (between 3 and 15 hours). Thus, we divided the distribution into two stages and estimated parameters for each (by eye). Figure 5 shows the observed distribution and the improved model. Clearly the two-stage model is a better fit.

Calculating the CLM for the multi-stage model is only slightly more complicated than for the single-stage model. Given the age of a job, we use Bayes' theorem to calculate p_i, the probability that the job belongs to each class. Then the conditional distribution is

$$\begin{aligned} cdf_{L|a} &= Pr\{L < t|L > a\} \\ &= \sum_i p_i \cdot Pr\{L < t|L > a, stage = i\} \end{aligned}$$

where i enumerates the stages of the model. For a given stage, we calculate the conditional distribution using Equation 3 with the parameters for that stage.

Using the improved model does not significantly improve the performance of either predictor. Thus we conclude that

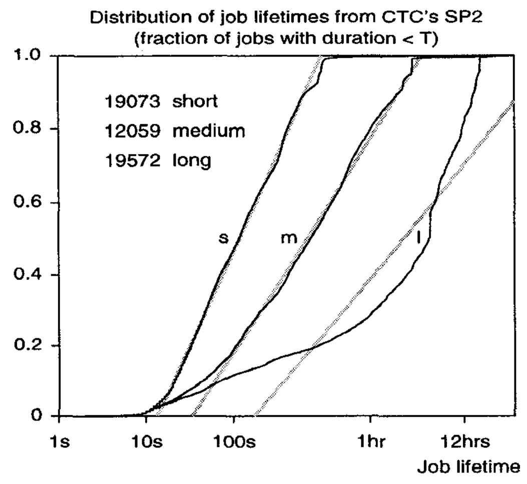


Figure 6. The distribution of lifetimes for batch jobs submitted to the IBM SP2 at CTC, broken down according to the queue to which they were submitted.

an ill-fitting model does not impair the predictors *per se*, but rather that the bimodal shape of the distribution is the root of the problem. A bimodal distribution indicates that there are two classes of jobs and that users are failing to distinguish between them. The multistage model, even if it fits the distribution well, does not recover this lost information, and therefore does not improve our predictions. The only way to improve the predictions is to solicit better information from users.

6 Another place, another time

Since we know no theoretical reason why the distribution of lifetimes should fit the uniform-log model we propose, it is natural to ask whether the techniques presented here are applicable to other environments.

To answer this question, we obtained data from the IBM SP2 at the Cornell Theory Center [9], including the submission times, execution times and cluster sizes for 50862 jobs submitted during the six-month interval from June 18 to December 2, 1995.

We divided these jobs into three groups — short, medium and long — according to the name of the queue to which they were submitted. Figure 6 shows the distribution of lifetimes for these groups. As with the jobs on the Paragon at SDSC, the uniform-log model fits well for the short and medium groups, and not as well for the long group. Nevertheless, the R^2 values for these fits are somewhat better across the board than those from the Paragon data:

Table 4

	β_0	β_1	R^2
short	-.6	.23	.999
medium	-.58	.17	.995
long	-.65	.13	.84

The relationships among the three curves are not the same as among the jobs from the Paragon. On the Paragon the estimated lines were nearly parallel; only the intercept varied from group to group. On the SP2, the lines have nearly the same intercept; it is their slopes that vary. This observation suggests that these distributions vary from environment to environment, but that the proposed model is able to span this range of behavior.

Next, we submitted the trace data from the SP2 to the same simulator we used for the Paragon data. The only change we made to the simulator was to plug in the estimated parameters from Table 4.

Of the 50862 jobs in the trace, 4026 received predictions from Predictor A and 4156 received predictions from Predictor B. The performance of the two predictors follows the same pattern as with the SDSC data — Predictor A is best for small values of n'; Predictor B for large. The CC for the combined predictor is 0.72, which is a significantly better than the predictions for the SDSC data ($CC = 0.65$).

One reason for this improvement is that the SP2 is a bigger machine (430 nodes *vs.* 368 on the Paragon), and the average cluster size of jobs tended to be smaller (9.2 *vs.* 25.2 on the Paragon). Thus, there tend to be more jobs in the system at any given time. Since our predictions are based on the aggregate behavior of many jobs, we expect them to improve as the number of jobs in the system increases. Another reason for the improvement is that users at CTC appear to be providing better run time estimates than users at SDSC: there is less overlap between the distributions from different queues, and the distribution of lifetimes for long jobs is less bimodal.

6.1 Another time

So far we have been cheating by using the trace data to estimate the distribution parameters, and then using the estimated parameters as inputs to the prediction algorithm. Of course, a real system would not have the luxury of knowing ahead of time the exact distribution of lifetimes of the jobs that would be submitted.

Assuming that the distributions do not change drastically over time, though, it should be possible to make accurate predictions using parameters estimated from recent accounting data. To test this claim, we divided our trace data into four intervals, each four months long. For each interval we found the distribution of lifetimes (broken down by queue

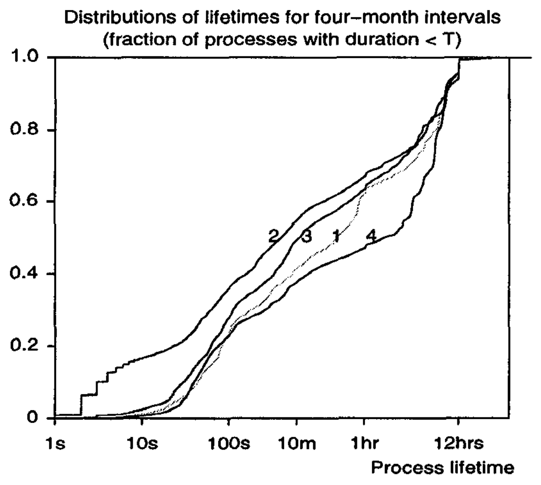


Figure 7. The distribution of lifetimes on the Intel Paragon at SDSC during four four-month intervals from January 1, 1995 to May 1, 1996.

name, as in Section 3.1) and estimated the parameters of each distribution.

Figure 7 shows the distribution of lifetimes for each interval. It is clear that this distribution changes over time, and that for the last year there has been a trend toward longer run times. Hotovy *et al.* [11] report a similar trend on the IBM SP2 at CTC — these trends may indicate a maturing workload as users submit fewer test runs and more production runs.

Table 5 shows the median[2] lifetime of jobs submitted to each queue (in seconds). The lifetimes of sequential jobs have been consistent over time, but short jobs have been getting shorter, and both medium and long jobs have been getting longer.

Table 5

	1–4/95	5–8/95	9–12/95	1–4/96
sequential	250	238	314	244
short	446	169	92	98
medium	903	1182	1023	5797
long	9209	14,161	9601	17,995
all jobs	1332	491	903	2250

To evaluate the impact of obsolete parameters on the accuracy of our predictors, we simulated the last three trace intervals (May '95 through April '96) using, in each interval, the parameters estimated during the *previous* interval. We feel that this is a realistic simulation of a system that updates its parameters every four months based on accounting data from the previous four months.

In general, the performance of the predictors is not as good as with the optimistically accurate parameters we have been using: CC for Predictor A is 0.60 (down from 0.63), for Predictor B is 0.57 (down from 0.61), and for the combined predictor is 0.62 (down from 0.65). But the difference in accuracy is not large, suggesting that the predictors are not greatly impaired if the parameters are obsolete. Thus, we expect that in a practical system it would be sufficient to estimate new parameters only occasionally (maybe monthly).

In this workload, the distribution of lifetimes for medium and long jobs is becoming increasingly bimodal (similar to Figure 5). This indicates that there are actually two classes of jobs in these queues, and that users are failing to distinguish between them when they choose a queue. As the quality of information provided by users declines, we find that our predictions become less accurate. We can mitigate this effect in part by using additional information about jobs to create more classes. For example, we found that both cluster size and requested memory size are correlated with run time. We could create a set of classes that is the cross product of the sets of attributes; the only limit on the number of classes is that we need enough jobs in each class to get a good estimate of the class' distribution.

Another way to improve the information content of the distributions is to change the interface to the queueing system to provide better feedback about the run times of users' jobs and thereby solicit better estimates.

7 Conclusions

- We have proposed a workload model for batch jobs on space-sharing parallel computers, based on a *uniform-log distribution* of lifetimes. We show that the proposed model captures the behavior of real workloads in two different environments. This model can be used to generate synthetic workloads for simulating and evaluating scheduling policies for similar environments.
- We have used this workload model to develop statistical techniques for predicting queue times for waiting jobs. The proposed techniques worked well in simulations of both of the environments we tested. These techniques should be useful for several applications, especially selection of cluster sizes on space-sharing parallel computers.
- Many scheduling policies for supercomputing environments depend on information provided by users about the resource requirements of their jobs. We observe that this information is often unreliable, but show that the proposed techniques are able to distill this information in a useful way.

[2]We calculated median values by estimating the parameters of each distribution and calculating the median of the model distribution. We have found that these estimates are more robust than conventional medians, but overall the values are similar and show the same trends.

7.1 Future work

In this paper we have presented algorithms for predicting the wait time of a job after it reaches the head of the queue. We have not addressed the question of how long it will take to get there. Predictor B can be extended in a natural way to predict the behavior of jobs in queue, and thus to predict the entire queue time for a new arrival. We have implemented this extension and seen promising preliminary results. Predictor B can also be extended to handle non-FCFS scheduling strategies, such as the priority scheme used at SDSC.

As part of the Distributed Object Computation Testbed (DOCT) project [13], we are implementing a persistent Legion [7] object that monitors the load on space-sharing parallel computers and answers queries regarding expected queue times. Application-Level Schedulers (AppLeS) will use this information for external resource selection [2].

Acknowledgements

We would like to thank George Kremenek at SDSC and Steven Hotovy at CTC for their help with the accounting data. We are also grateful to Paul Hilfinger at U.C. Berkeley, Reagan Moore at SDSC, Jennifer Schopf and Rich Wolski at U.C. San Diego, and Emilia Rosti at the University of Milan for reading drafts of this paper and providing valuable comments.

References

[1] M. J. Atallah, C. L. Black, D. C. Marinescu, H. J. Siegel, and T. L. Casavant. Models and algorithms for coscheduling compute-intensive tasks on a network of workstations. *Journal of Parallel and Distributed Computing*, 16(4):319–327, Dec 1992.

[2] F. Berman and R. Wolski. AppLeS home page http://www-cse.ucsd.edu/groups/hpcl/apples/apples.html. University of California at San Diego, 1996.

[3] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement-based study on UNIX. *IEEE Transactions on Software Engineering*, 15(12):1579–86, December 1989.

[4] A. B. Downey. A model for speedup of parallel programs. Technical report, University of California at Berkeley, 1997.

[5] A. B. Downey. Using queue time predictions for processor allocation. Technical Report CSD-97-929, University of California at Berkeley, 1997.

[6] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 215–227, 1995.

[7] J. C. French, A. S. Grimshaw, W. A. Wulf, A. C. Weaver, and P. F. Reynolds, Jr. Legion home page http://www.cs.virginia.edu/~mentat/legion/legion.html. University of Virginia, 1996.

[8] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 13–24, May 1996.

[9] S. Hotovy. Personal correspondence. 1996.

[10] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In *IPPS '96 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 15–22, 1996.

[11] S. Hotovy, D. Schneider, and T. O'Donnell. Analysis of the early workload on the Cornell Theory Center IBM SP2. Technical Report 96TR234, Cornell Theory Center, January 1996.

[12] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM Sigmetrics*, volume 14, pages 54–69, 1986.

[13] R. Moore and R. Klobuchar. DOCT home page http://www.sdsc.edu/DOCT. San Diego Supercomputer Center, 1996.

[14] K. Steube and R. Moore. Paragon throughput analysis. In *Proceedings of the Intel Supercomputer Users Group*, pages 264–267, June 1994.

[15] M. Wan, R. Moore, G. Kremenek, and K. Steube. A batch scheduler for the Intel Paragon with a non-contiguous node allocation algorithm. In *IPPS '96 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 29–40, 1996.

[16] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In *6th Symposium on the Frontiers of Massively Parallel Computation*, 1996.