Predicting Bounds on Queuing Delay for Batch-scheduled Parallel Machines

John Brevik, Daniel Nurmi, and Rich Wolski[*]
Computer Science Department
University of California, Santa Barbara
Santa Barbara, California 93106

Abstract

Most space-sharing parallel computers presently operated by high-performance computing centers use batch-queuing systems to manage processor allocation. In many cases, users wishing to use these batch-queued resources have accounts at multiple sites and have the option of choosing at which site or sites to submit a parallel job. In such a situation, the amount of time a user's job will wait in any one batch queue can significantly impact the overall time a user waits from job submission to job completion. In this work, we explore a new method for providing end-users with predictions for the bounds on the queuing delay individual jobs will experience. We evaluate this method using batch scheduler logs for distributed-memory parallel machines that cover a 9-year period at 7 large HPC centers.

Our results show that it is possible to predict delay bounds reliably for jobs in different queues, and for jobs requesting different ranges of processor counts. Using this information, scientific application developers can intelligently decide where to submit their parallel codes in order to minimize overall turnaround time.

1. Introduction

Typically, high-performance multi-processor compute resources are managed using *space sharing*, a scheduling strategy in which each program is allocated a dedicated set of processors for the duration of its execution. In production computing settings, users prefer space sharing to time sharing, since dedicated processor access isolates program execution performance from the effects of a competitive load. Because processes within a partition do not compete for CPU or memory resources, they avoid the cache and translation look-aside buffer (TLB) pollution effects that time slicing can induce. Additionally, inter-process communication occurs with minimal overhead, since a receiving process can never be preempted by a competing program.

For similar reasons, resource owners and administrators prefer space sharing as well. As long as the time to allocate partitions to, and reclaim partitions from, parallel programs is small, no compute cycles are lost to time-sharing overheads, and resources run with maximal efficiency. Thus, at present, almost all production high-performance computing (HPC) installations use some form of space sharing to manage their multi-processor and cluster machines.

[*]This work was supported by grants from the National Science Foundation numbered CCF-0331654 and NGS-0305390, and by the San Diego Supercomputer Center.

Because each program in a space-shared environment runs in its own dedicated partition of the target machine, a program cannot be initiated until there are a sufficient number of processors available for it to use. When a program must wait before it can be initiated, it is queued as a "job" [1] along with a description of any parameters and environmental inputs (*e.g.* input files, shell environment variables) it will require to run. However, because of the need both to assign different priorities to users and to improve the overall efficiency of the resource, most installations do not use a simple first-come-first-served (FCFS) queuing discipline to manage the queue of waiting jobs. Indeed, a number of queue-management systems, including PBS [28], LoadLeveler [1], EASY [19], NQS/NQE [23], Maui [21] and GridEngine [16] each offers a rich and sophisticated set of configuration options that allow system administrators to implement highly customized priority mechanisms.

Unfortunately, while these mechanisms can be used to balance the need for high job throughput (which ensures machine efficiency) with the desires of end-users for rapid turnaround times, the interaction between offered workload and local queuing discipline makes the amount of time a given job will wait highly variable and difficult to predict. Users may wait a long time – considerably longer the the job's eventual execution time – for a job to begin executing. Many users find this potential for unpredictable queuing delay particularly frustrating since, in production settings, they *can* make fairly reliable predictions of how long a program will execute once it starts running. Without an ability to predict its queue waiting time, however, users cannot plan reliably to have results by a specific point in time.

In this paper, we present the *Binomial Method Batch Predictor* (BMBP) – a new methodology for predicting bounds, with quantitative confidence levels, on the amount of time an individual job will wait in queue before it is initiated for execution on a production "batch scheduled" resource. BMBP bases its predictions only one the observed history of previous waiting times. Thus, it automatically takes into account the effects of varying workload and customized local queuing discipline. In addition, we observe that the queuing behavior exhibited by all of the machines we examined in this study (7 supercomputers operated by the National Science Foundation and the Department of Energy over a 9-year period) is highly variable. In response to hardware and software upgrades, failures, and configuration changes, changing organizational priorities, user turnover, security events, *etc.*, administrators appear to tune and adjust their local queuing policies, often in a way that is not obvious to the user community. BMBP attempts to detect these *change points* adaptively so that it uses only relevant

[1]We will use the term "job" throughout this paper to refer to a description of a program and its execution requirements that a queuing system can use to initiate a program once the necessary resource become available.

history to make each prediction.

We verify both the efficacy and generality of BMBP using the logging information recorded by various batch schedulers that were in use during the time each machine in our study was in operation. All of the installations except the Lawrence Livermore National Laboratory maintained a variety of queues for each machine. We presume that a qualitative queuing policy has been published to the user community for each queue (*e.g.*, jobs in the "Low" queue at the San Diego Supercomputer Center would be given lower priority than those in the "Normal" queue, which would, in turn, have lower priority than those in the "High" queue). In this way these installations attempt to provide their respective users communities with a rudimentary and qualitative prediction capability since, in general, lower-priority jobs can be expected to wait longer in queue.

However, in each case the batch scheduler must choose among jobs that are waiting in a number of queues, each of which is governed by a specific policy. Moreover, the algorithm used to select a particular job at a particular time from amongst the various queues is not typically published and can potentially change under administrator control. Thus, while the implementation of multiple policies for a given machine through multiple queues can provide a high-level, qualitative expectation of how a specific job will be treated, it substantially complicates the problem of making a quantitative prediction for that job's wait time.

We examine the predictive power of BMBP when it is applied to the various queues implemented at each site by detailing how well our new method predicts in a quantitative way the qualitative characteristics attached to each queue. With implicit priority mechanisms such as backfilling [18] in use at some of the sites, however, users have come to expect that processor count also affects wait time. In particular, jobs in a particular queue requesting small numbers of processors are believed, typically, to wait for shorter periods, since they can be "backfilled" into the machine around larger jobs. We therefore also examine how well BMBP predicts the bounds on waiting times for jobs based on the queue to which they were submitted and the number of processors they specified. In all cases – covering over 1 million jobs – the method makes predictions *for each job*, which are "correct" in a very specific statistical sense which we will discuss below, for the bounds on the waiting time.

This ability to make predictions for individual jobs distinguishes our work from other previous efforts. An extensive body of research [29, 7, 8, 11, 14, 5, 10, 12] investigates the statistical properties of offered job workload for various HPC systems. By providing a rigorous statistical characterization of job interarrival times and program execution times, the resulting statistical properties associated with queuing time can be derived through simulation. Despite these extensive characterization studies, however, we know of few previous research efforts that treat the problem of predicting queuing delay in a quantitative way. We emphasize, however, that our goal is strictly to provide a predictive mechanism for users and application schedulers rather than to investigate the distributional properties exhibited by HPC systems. In particular, BMBP makes a prediction for each individual job's queuing delay rather than a statistical characterization of the queuing delay experienced by a set of jobs (e.g. using an estimate of the mean waiting time).

The remainder of this paper details BMBP and describes its evaluation. In so doing, the paper makes the following two novel contributions.

- We describe a new predictive methodology for bounding queuing delay that is quantitative, non-parametric, and general. As a result, the method works automatically, without ancillary analysis or human "tuning" for a specific site or a specific queue.

- We evaluate this methodology by comparing its performance to an alternative parametric approach based on the assumption that the underlying distribution is either log-normal or Weibull. Our results show that our new approach achieves both the desired confidence levels and the tightest bounds (in aggregate) for the cases under study.

We have developed BMBP to provide a practically realizable predictive capability for deployment as a user and scheduling tool. Therefore our reportage focuses on the results generated by a working prototype that is currently undergoing integration with various batch scheduling systems, and our results are, ultimately, empirical.

2. Related Work

Smith, Taylor, and Foster in [29] use a template-based approach to categorize and then predict job execution times. From these execution-time predictions, they then derive queue delay predictions by simulating the future behavior of the batch scheduler in faster-than-real time. Our work differs from this approach in two significant ways. To be effective, the Smith-Foster-Taylor method depends both on the ability to predict job execution time accurately for each job and on explicit knowledge of the scheduling algorithm used by the batch scheduler. Other work [17, 6] suggests that making such predictions may be difficult for large-scale production computing centers. Moreover, the exact details of the scheduling policy implemented at any specific site is typically unpublished. While the algorithm may be known, the specific instance of the algorithm and the definition of any parameters it requires are the prerogative of the site administrators and, indeed, may be changed as conditions and site-specific needs warrant. In contrast, our approach uses only the observed queue delays. By doing so, it does not require execution-time predictions, and it automatically takes into account any site-specific effects induced by the local scheduling policy (whether static or dynamically changing).

Downey [7, 8] uses the log-uniform distribution to model the remaining lifetimes of jobs executing in all machine partitions as a way of predicting when a "cluster" of a given size will become available and thus when the job waiting at the head of the queue will start. Our work differs from Downey's in that we do not use predictions of the time until resources become free to estimate the start time of a job. Rather, we work directly from the observed queuing delays.

Finally, our approach differs from both of these related approaches in that it attempts to establish rigorous bounds on the time an individual job will wait rather than a specific, single-valued prediction of its waiting time. We contend that the highly variable nature of observed queue delay is better represented to potential system users as quantified confidence bounds than as a specific prediction, since users can "know" the probability that their job will fall outside the range.

3. Problem Definition: Predicting Bounds on Queuing Delay

If we are to treat the problem of predicting queuing delay statistically, the best possible outcome (from the job submitter's perspective) is the ability to predict bounds on the delay a job will experience, and to do so with a quantifiable measure of confidence. However, much of the observed queue delay data is highly skewed making moment-based descriptive statistics such as mean and standard deviation potentially deceptive. In such cases, order statistics (such as median and quartiles) are generally considered more appropriate (*cf.* [22], chapter 1, section 2). For example, knowing that the *mean* wait time a user can expect is 24 hours is likely to be less useful than knowing that there is a 75% chance that the wait time will be less than 15 minutes – which is not an unrealistic state of affairs for a batch queue.

Now, suppose that a scheduler or machine user would like to know the maximum amount of time a job is likely to wait in a batch queue before it is executed. In order to be precise, we quantify the word "likely" to mean that we wish to generate a predicted number of seconds so that we are 95% certain that our job will begin execution within that number of seconds, in the sense that, over time, 95% of our predictions will be at least as great as the actual wait-times of the jobs. If we regard the wait time of a given job as a random variable, then, this amounts to finding an estimate for the 95^{th} percentile, or 0.95 *quantile*, of this variable's distribution. Note that an estimate of the mean and standard deviation provides little predictive information for this question.

Since the distribution of interest is unknown, any of its parameters in which we might be interested must be estimated, typically from a sample. Standard methods of statistical inference allow us to use a sample to produce an interval (which may be infinite on one end) that we can assert contain the parameter with a specified level of *confidence*, roughly corresponding to the "probability" that our interval has captured the true parameter of the population. In general, the more confident we wish to be, the wider the confidence range; for example, a 99% confidence interval for the estimated 0.95 quantile is wider than an 80% confidence interval, because the higher level of confidence demands that we be more certain that the true parameter lies in our interval. For the purposes of this paper, we will typically be considering upper confidence *bounds* on quantiles, which correspond to left-infinite intervals $(-\infty, B]$.

To estimate an upper bound, then, we need to choose two values: the quantile and the desired level of confidence for the bound. Returning to the example, to say that a particular statistical method produces a 99%-confidence upper bound on the 0.95 quantile is to say that, if the method is applied a large number of times, the value it produces fails to be greater than the 0.95 quantile no more than 1% of the time. We will term an upper-bound prediction as *correct* if the observed value falls below the predicted value; we will term a prediction method on a set of data *correct* if the proportion of correct predictions it makes is at least as great as the quantile it is predicting.

In this work, we have chosen to use the value 0.95 for each. We have identified the 0.95 quantile as appropriate for a level of how certain we wish to be about how long a job will wait in the queue. At the same time, 95% is fairly standard from the standpoint of statistical inference as a level of confidence. Note that because it is the 0.95 quantile we are estimating, a user should expect that there is at most a 1 in 20 chance that the actual wait time experienced by a job exceed the predicted wait time (provided, of course, that the prediction method is correct in the sense of the above paragraph).

Our aim in producing predictions is not only that they be correct at least 95% of the time, but also that they be meaningful to the user. If we were to make extremely conservative predictions, based, say, on the maximum wait time ever observed in the queue, the percentage of correct predictions would doubtless increase; however, the extremely large predictions produced would have little utility to someone wishing to use these values for planning purposes. One sees, then, that there is a trade-off between having a high percentage of correct predictions and those predictions reflecting what a "typical" wait time might be: If the predictions are correct at a substantially higher rate than advertised, it is a sign that they are overly conservative and therefore less meaningful than they could be. Thus the fact that, in general, only slightly more than 95% of our predictions are correct for each queue, as we will see in Section 6, shows that they are meaningful for the purpose for which they are designed.

Note also that, while we have presented the problem in terms of estimating an upper bound on queuing delay, it can be similarly formulated to produce lower confidence bounds, or two-sided confidence intervals, at any desired level of confidence. It can also be used, of course, for any population quantile. For example, while we have focused in this paper on the relative certainty provided by the $.95$ quantile, our method estimates confidence bounds for the median (*i.e.*, the point of "50-50" probability) with equal effectiveness. We note that the quantiles at the tail of the distribution corresponding to rarely occurring but large values are more variable, hence more difficult to estimate, than those nearer the center of the distribution. Thus, in a typical batch queue setting, which is characterized by large numbers of jobs experiencing short wait times and a few jobs experiencing long wait times, the upper quantiles provide the greatest challenge for a prediction method. By focusing on an upper bound for the $.95$ quantile, we are testing the limits of what can be predicted for queue delay.

4. Inference for Quantiles

In this section, we describe our approach to the problem of determining upper bounds, at a fixed level of confidence, for quantiles of a given population whose distribution is unknown. As described previously, our intention is to use this upper bound as a conservative estimate for the queuing delay, and to report the degree of conservatism as the quantified confidence level.

4.1 The Binomial Method Batch Predictor

Our approach, which we term the *Binomial Method Batch Predictor* (BMBP), is based on the following simple observation: If X is a random variable, and X_q is the q quantile of the distribution of X, then a single observation x from X will be greater than X_q with probability $(1 - q)$. (For our application, if we regard the wait time, in seconds, of a particular job submitted to a queue as a random variable X, the probability that it will wait for less than $X_{.95}$ seconds is exactly $.95$.)

Thus (provisionally under the typical assumptions of independence and identical distribution) we can regard all of the observations as a sequence of independent Bernoulli trials with probability of success equal to q, where an observation is regarded as a "success" if it is less than X_q. If there are n observations, the probability of exactly k "successes" is described by a Binomial distribution with parameters q and n. Therefore, the probability that more than k observations are greater than X_q is equal to

$$1 - \sum_{j=0}^{k} \binom{n}{j} \cdot (1-q)^j \cdot q^{n-j} \qquad (1)$$

Now, if we find the smallest value of k for which Equation 1 is larger than some specified confidence level C, then we can assert that we are confident at level C that the k^{th} value in a sorted set of n observations will be greater than or equal to the X_q quantile of the underlying population – in other words, the k^{th} sorted value provides a *level-C upper confidence bound* for X_q.

Clearly, as a practical matter, neither the assumption of independence nor that of identical distribution (stationarity as a time series) hold true for observed sequences of job waiting times from the real systems, and these failures present distinct potential difficulties for our method. In the remainder of this section, we address the statistical characteristics of these difficulties. Section 6 demonstrates the effectiveness of (and, as a result, the impact of these assumptions on) BMBP when applied to batch queues *in situ*.

Let us first address the issue of independence, assuming for the moment that our series is stationary but that there may be some autocorrelation structure in the data. We hypothesize that the time-series process associated to our data is *ergodic*, which roughly amounts to saying that all the salient sample statistics asymptotically approach the corresponding population parameters. Ergodicity is a typical and standard assumption for real-world data sets; *cf., e.g.,*[15]. Under this hypothesis, a given sample-based method of inference will, *in the long run*, provide accurate confidence bounds[2] The high numbers of jobs at large-scale centers allow the long-run nature of the bounds produce by BMBP to take effect and produce the desired rate of success.

Although our method is not invalidated by dependence in the series of measurements we examine to make a prediction, we do not claim that our method yields uncorrelated errors, which is a typical and desirable property for time-series-based estimation; rather, our method produces bounds that will in the long run be correct the desired fraction of the time, even when the data has complex correlation structure, as long as the series is stationary.

[2]As an example, imagine tossing a coin which has the strange property that, if a head is tossed, the probability that the next toss will be a head is $.9$, and likewise, if a tail is tossed, the probability that the next toss will be a tail is $.9$. The proportion of heads we will see in the first few tosses is quite variable and depends strongly on the first toss, and the tosses are not independent; nevertheless, in the long run the proportion of heads will converge to $.5$.

A separate issue from the *validity* of our method is that exploiting any autocorrelation structure in the time series should, *in principle*, produce more accurate predictions than a Binomial Method which ignores these effects. Indeed, most time-series analysis and modeling techniques are primarily focused on using dependence between measurements to improve forecasting [3]. For the present application, however, there are a number of confounding factors that foil time-series methods. First of all, for a given job entering a queue, there are typically several jobs in the queue, so that the most recent available wait-time measurement is for several time-lags ahead. The correlation between the most recent measurement at the time a job enters the queue and that job's eventual wait time is typically modest, around 0.1, and does not reliably contribute to the accuracy of wait-time predictions. Another issue is the complexity of the underlying distribution of wait times: They typically have more weight in their tails than exponential distributions, and many queues exhibit bimodal or multimodal tendencies as well. All of this makes any linear analysis of data relationships (which is the basis of the "classical" time-series approach) very difficult. Thus while the data is not independent, it is also not amenable to a standard time-series approach for exploiting correlation.

Non-stationarity and Changepoint Analysis

Unlike the issue of independence and correlation, the issue of non-stationarity *does* place limitations on the applicability of our method. Clearly, for example, it will fail in the face of data with a "trend," for example a mean value that increases linearly with time. On the other hand, insisting that the data be stationary is too restrictive to be realistic: Large compute centers change their scheduling policies to meet new demands, new user communities migrate to or from a particular machine, *etc.* It seems to be generally true across the spectrum of traces we have examined (described in Section 5.2) that wait-time data is typically stationary for a relatively long period and then undergoes a "changepoint" into another stationary regime with different population characteristics. We thus present the BMBP as a prediction method for data which are stationary for periods and for which the underlying distribution changes suddenly and relatively infrequently; we next discuss the problem of detecting changepoints in this setting.

The problem of changepoint analysis in time series typically focuses on the case when a family of models is specified and the data are analyzed *a posteriori* for points at which the parameters of the model change. This outlook is unsatisfactory for our purposes for at least two reasons. First, we adopted our non-parametric approach to the batch-queue problem precisely in order to avoid specifying a model for the data, which may typically exhibit multimodal behavior (even during a stationary regime) and is resistant to accurate parametric modeling. Second, we wish to detect changepoints on the fly, as rapidly as possible, in order to avoid long periods of wildly inaccurate predictions.

We address the problem of finding changepoints in the following way. Given an independent sequence of data from a random variable X, we deem that the occurrence of three values in a row above $X_{.95}$ constitutes a "rare event" and one which should be taken to signify a changepoint. Why three in a row? To borrow a well-known expression from Tukey [3], two is not enough and four is too many; this comes from consideration of "Type I" error. Under the hypothesis of identical distribution, a string of two consecutive high or low values occurs every 400 values in a time series, which is an unacceptable frequency for false positives. Three in a row will occur every 8000 values; this strikes a balance between sensitivity to a change in the underlying distribution of the population and certainty that a change is not being falsely reported.

Now, suppose that the data, regarded as a time series, exhibits some autocorrelation structure. If the lag-1 autocorrelation is fairly strong, three or even five measurements in a row above the $.95$ quantile might not be such a rare occurrence, since, for example, one unusually high value makes it more likely that the next value will also be high. In order to determine the number of consecutive high values (top 5% of the population) that constitute a "rare event" approximately in line with the criterion spelled out for independent sequences, we conducted a Monte Carlo simulation with various levels of lag-1 autocorrelation in $AR(1)$ time series [15], observed the frequencies of occurrences of consecutive high and low values, and generated a lookup table for rare-event thresholds.

For a given set of data, then, we use the autocorrelation structure from its "training period" to determine its rare-event threshold, and then periodically update the threshold as new data appears. There is one subtlety to note, however: In keeping with the spirit of non-parametric treatment of the data and in order to make direct comparisons as much as possible, we "normalize" the measurements of each data set to have underlying distribution $N(0, 1)$. We also suspected (and have subsequently confirmed) that the normalization process has the effect of linearizing some of the time-series characteristic of the data, so that the autocorrelations become somewhat stronger. Note that this process is harmless for our method of inference, since it is invariant under any order-preserving transform of the data.

When we observe the determined number of consecutive incorrect predictions that constitute a rare event, we assume that the data has changed in some fundamental way so that old data is no longer relevant for our predictions. Accordingly, we trim the history as much as we are able to while still producing meaningful confidence bounds.

For example, it follows from formula 1 above that in order to produce a 95% confidence bound for the $.95$ quantile the minimum history from which a statistically meaningful inference can be drawn is 59: Set $j = 0$, so that the sum gives the probability that at least one is more than X_q; the smallest n for which this sum is at least $.95$ is 59. Therefore, for this specific quantile and level of confidence, upon seeing the assigned number of missed predictions in a row (determined by the first autocorrelation observed during training), we would trim our history to the most recent 59 (so that we can at least make some sort of predictions) and start making predictions based on the shortened history, keeping the history "window" at 59 until our history lies entirely after the detected changepoint. Thus our method automatically adapts to the longest history that is clearly relevant to the current prediction.

For the data sets considered, our method produces (conservative) predictions for the $.95$ quantile for each wait time so that, for each data set, our predictions are correct at least 95% of the time. The rare-event detection method is effective in handling changepoints (although prediction errors near changepoints tended to be more frequent) , and the relatively high level of confidence chosen enables the predictor to work well in spite of possible effects of short-term non-stationarity in the data.

4.2 Model-Fitting Using Log-Normal and Weibull Distributions

In [7], Downey hypothesizes that the job at the head of a FCFS queue experiences a delay that is well-modeled by a *log-uniform* distribution. In a private communication with the author, he expressed a belief that overall wait times are well modeled by *log-normal* distributions; recall that that the distribution of a random variable X is log-normal if $\log X$ is normally distributed. This observation suggests another approach to the problem of producing quantile estimates for batch-queue wait times; specifically, one can fit a parametric distribution to the data using, preferably, the method of maximum likelihood estimation (MLE) [20], and then produce the desired population quantile from a lookup table or the inverse of the cumulative distribution function. Our previous experiences with predicting process lifetime durations [25, 4, 26] and visual inspection of the data suggest that the Weibull distribution might also serve as a good model for wait-time data. Thus we compare BMBP to predictions generated from log-normal and Weibull models.

In order to make a strict comparison between model-based methods and BMBP, it may be pointed out that it would be necessary, rather than generating estimates using MLE, to produce an upper confidence bound on the estimates. In fact, we found that the MLE-based estimates already tend to be conservative, and this conservatism would only be exacerbated with upper confidence bounds, so we choose to report the model-based method that performs the best according to our criteria. Moreover, computing confidence bounds for quantiles assuming a Weibull model requires significant computational effort. This effort does not seem warranted, since it would only serve to make the predictions even more conservative.

Initially, we implemented our model-based predictors to use the full history of available measurements in each case. However, in light of the long-term non-stationarity phenomenon discussed above, incorporating the same history-truncation strategy that we use with BMBP improves the performance of the model-based approaches as well. Indeed, we observed a substantial improvement in both correctness and accuracy in both model-based techniques when we incorporated our changepoint detector. Therefore, in this study, the log-normal and Weibull parametric approaches we investigate use the same history-trimming methodology as does BMBP.

[3] We refer here to Tukey's notorious explanation why the "whiskers" in a boxplot should extend 1.5 IQRs, namely that "1 is too small and 2 is too large"; beyond its beautiful "sound bite" quality, Tukey's quote serves as a reminder that any statistical threshold, such as 95% confidence or $.05$ significance level, is an artificial entity ultimately chosen for its usefulness.

5. Evaluation

Our goal is both to determine the statistical correctness of BMBP and to investigate its accuracy. Recall that a method is correct if, provided the number of job predictions is large enough to offset short-term statistical anomalies, the percentage of correct predictions is at least as large as the specified quantile. While we have examined several different combinations of quantile and confidence level as part of our research, for the purpose of brevity, we report only on a single combination – the 95th percentile with confidence level 0.95 – in this work. As a measure of accuracy, we detail the degree of over-prediction each upper bound generates as the square root of the mean square over-prediction error. That is, in the cases when BMBP and the other tested methodologies correctly produce a success percentage greater than 95%, we wish to detail how "tight" (in aggregate) the successful predictions are. For example, notice that a simple prediction method in which the predictor repeatedly guesses an astronomically large number 19 times followed by a single guess of a very small number will generate predictions that are above the corresponding observations *exactly* 95% of the time and therefore, under our definitions, is "correct." On the other hand, it is not an "accurate" predictor, in a way that we will discuss.

While we have deployed BMBP in production computing settings (*cf.* Subsection 6.3), to first determine its efficacy, we use a trace-based event-driven simulation (described in the next subsection). Logging data from a variety of HPC sites (described in Subsection 5.2) records the queue name, arrival time, queue delay, and processor count for all of the jobs submitted to each system. Because we can replay each submission trace we can compare BMBP to alternative approaches based on a dynamically fit log-normal and Weibull distributions determined by an MLE over the same job workloads. For each job in each trace we record the prediction that the job's user *would have been given* if the particular method under test were in place when the job was submitted. However, since users might change their submission decisions based on the predictions furnished, this comparison only demonstrates that the method retroactively captures the dynamics that were present at the time of each submission.

We have also been able to obtain preliminary timings for BMBP from its use in simulation. Using a 1-gigahertz Pentium III, the average time required to make a prediction over the approximately 1.1 million predictions we examine across all batch queue logs is 8 milliseconds. Clearly BMBP is efficient enough to deliver timely forecasts.

5.1 Simulation Implementation

Our simulator takes as input a file containing historical batch-queue job wait times from a variety of machines/queue combinations and parameters directing the behavior of our models. For each machine/queue for which we have historical information, we were able to create parsed data files which contain one job entry per line comprising the UNIX time stamp when the job was submitted and the duration of time the job stayed in the queue before executing.

The steady state operation of the simulation reads in a line from the data file, makes a prediction based on the current model being used, and stores the job in a "pending queue". We then increment a virtual clock until one of two events occur.

- The virtual time specified for the job to wait in the pending queue expires.
- A new job enters the system according to the virtual clock.

When the first case occurs, the job is simply added to a growing list of historical job wait times that are available for forecasting. Although the waiting time for the job is carried in the trace, the predictor is only entitled to "see" the waiting times for jobs that have completed their waiting periods.

When the second case occurs, the current prediction value is used to make a prediction for the job entering the queue, the simulation checks to see if the predicted time for that job is greater than or equal to the actual time the job will spend in the pending queue (success), or the predicted time was less than the actual job wait time (failure). The success or failure is recorded, and the job is placed on the pending queue. Note that in a "live" setting this success or failure can only be determined after the job completed its waiting period.

We also arbitrarily discard the first 30 days in all traces. In developing our prediction methodology, we noticed that in a great many of the traces, the initial values were substantially different than the remainder of the trace. Moreover, trend behavior (discussed in Section 6 below) is prevalent in these initial periods. We speculate that the introduction of a new machine or new queue on an existing machine typically initiates a "burn-in" period during which users and administrators tune the queue's control policy and priority. During the burn-in, we observe long blockages followed by sudden "waves" of released jobs. We attribute this initial burstiness to administrator intervention in response to the recognition (perhaps at the hands of an irate user community) of an unforeseen policy consequence. For the most part (all but two of the traces) this initial period lasts no more than 30 days, however. We will further discuss the "burn-in" feature of our data sets in the results section.

5.2 Batch Queue Data

We obtained 7 archival batch-queue logs from different high-performance production computing settings covering different machine generations and time periods. From each log, we extracted data for the various queues implemented by each site. For all systems except the ASCI Blue Pacific system at Lawrence Livermore National Laboratory (LLNL), each queue determines, in part, the priority of the jobs submitted to it. For example, jobs submitted to the *interactive* queue at the National Energy Research Science Center (NERSC) are presumably given higher-priority access to available processors than those submitted to the *regularlong* queue in an effort to provide interactive users with shorter queuing delays.

Typically, a center publishes a set of constraints that will be imposed on all jobs submitted to a particular queue. These constraints include maximum allowable run time, maximum allowable memory footprint, and maximum processor count which the batch-queue software enforces. The priority mechanism used by the scheduler to select jobs from across the advertised queues, however, is either partially or completely hidden from the user community and may change over time. For example, the center may choose temporarily to give higher priority to long-running large jobs immediately before a site review or nationally visible demonstration. While the user community may be informed of the change and its duration, they may not be told exactly how it will affect the priority given to jobs submitted to other queues.

Typically, however, centers do provide qualitative guidance regarding the priorities given to jobs as a function of their processor count. Large jobs (with high processor counts) may either be encouraged (as at the San Diego Supercomputer Center) by enjoying a higher priority, or given a lower priority in an effort to improve throughput. To capture these differences, we further subdivide the data in each queue according to the number of processors requested by each job. The processor ranges we use are 1-4, 5-16, 17-64 and 65+ [4]. Because subdividing the logging data reduces the number (and potentially the frequency) of jobs considered by each method, we discard any case for which the total number of jobs available is less than 1000. Since each of the logs spans a year or more, we believe it will be difficult to achieve significant results when fewer than 4 jobs per day, on the average, of a particular node count are submitted.

For this study, we consider trace data composed of 1.1 million jobs covering 9 years of operation at National Science Foundation and Department of Energy "open" computing centers. In particular, we consider job submission data from three machines operated by the San Diego Supercomputer Center during three different periods: the Intel Paragon (January 1996 to January 1997), the IBM SP-2 (April 1998 to April 2000), and the IBM Power-4 system (Datastar, April 2004 to April 2005). We also use traces from the Los Alamos National Laboratory's (LANL's) SGI Origin 2000 (December 1999 to April 2000), Lawrence Livermore National Laboratory's (LLNL's) SP-2 (Blue Pacific, January 2002 to October 2002), the SP-2 located at the National Energy Research Center (NERSC) at Lawrence Berkeley Laboratory (LBL, March 2001 to March 2003), and the Cray-Dell cluster operated by the Texas Advanced Computing Center (TACC, January 2004, March 2005). The LANL, SDSC SP-2 traces are available from Dror Feitelson's workload web site [9]. The Paragon trace is courtesy of Allen Downey, the NERSC trace is from Leonid Oliker at LBL, LLNL trace is from Brent Gorda at LLNL, and an initial TACC trace comes from Karl Schulz at TACC. In providing access to such detailed data, we cannot overstate the contribution these people have made to this work. Both the Datastar and eventual TACC traces we use were gathered using a Network Weather Service [27, 30] automatic sensor and predictor we developed for this project (*cf.* Section 6.3).

6. Results

In this section we investigate the efficacy of various methods for predicting queue delay quantiles with a quantified level of confidence. The simulation results are intended to describe the actual results a "live" prediction system would have generated if it had been available during the time epoch described by each trace under the assumption that the availability of these predictions would not affect submission or execution times. While it appears from our simulations that it is indeed possible to provide reliable estimates of the bounds on the delay quantile – and to do so in a way that takes into account the non-stationary nature of each series – there is considerable variation among the various methodologies we tested in terms of their accuracy and computational cost.

[4]These processor ranges were suggested to us by Jay Boisseau, Karl Schulz, and Warren Smith of the Texas Advanced Computing Center (TACC) based on experience with the TACC user community.

Machine	Queue	1-4	5-16	17-64	65+
datastar	TGhigh	B,W,L	-	-	-
datastar	TGnormal	B,W	-	-	-
datastar	express	B,W	B,W	-	-
datastar	high	none	B,W,L	-	-
datastar	normal	B,W,L	B,W,L	B,W,L	-
datastar	normal32	B,W,L	-	-	-
lanl	mediumd	-	-	-	B,W,L
lanl	short	-	-	B,W	-
lanl	chammpq	B,W,L	-	B,W,L	-
lanl	small	none	B,W,L	B,W,L	B,W
lanl	shared	B,W	B,W,L	-	-
lanl	scavenger	B,W,L	B,W	B,W	B,W,L
llnl	all	B,W	B,W	B,W	-
nersc	debug	B,W,L	B,W,L	-	-
nersc	low	B,W,L	W,L	B,W,L	-
nersc	premium	B,W	B,W,L	-	-
nersc	regular	B,W,L	B,W,L	none	-
nersc	reglong	B,W,L	-	-	-
sdsc	normal	B,W	B,W,L	B,W,L	-
sdsc	high	B,W,L	B,W,L	B,W,L	-
sdsc	low	B,W,L	B,W,L	B,W,L	-
sdsc	express	B,W	-	-	-
tacc2	normal	B,W,L	B,W,L	B,W,L	B,W
tacc2	devel	B,W,L	B,W,L	-	-
tacc2	serial	B,W	-	-	-

Table 1. BMBP simulation results indicating percentage of correct job wait time predictions.

6.1 Predicting By Queue Name and Processor Count

With scheduling improvements such as backfilling [18], dynamically changing user priorities (often at the behest of besieged system administrators or center personnel struggling to meet the requirements of an important demonstration), and statically defined priorities based on job size, users of modern batch systems have come to expect that processor count affects queuing delay. Thus, a common user desire is to be able to predict, at any point in time, an upper bound on delay for potential job submissions of different job sizes in a single queue.

To explore our ability to meet this need, we subdivide the jobs in each queue according to the number of processors specified in each submission request. Each subdivision corresponds to a range of processor counts as discussed in Section 5.2.

Machine	Queue	1-4	5-16	17-64	65+
datastar	TGhigh	1.32,2.20	-	-	-
datastar	TGnormal	2.93,F	-	-	-
datastar	express	2.07,F	1.53,F	-	-
datastar	high	fail	1.62,2.27	-	-
datastar	normal	2.45,2.82	3.16,2.89	1.60,1.87	-
datastar	normal32	3.23,7.92	-	-	-
lanl	mediumd	-	-	-	1.04,0.56
lanl	short	-	-	2.02,F	-
lanl	chammpq	0.28,0.08	-	1.80,4.77	-
lanl	small	fail	1.45,4.27	2.55,3.45	0.46,F
lanl	shared	1.38,F	0.31,0.10	-	-
lanl	scavenger	1.01,0.97	0.71,F	2.90,F	1.51,1.68
llnl	all	1.32,2.25	3.43,4.65	1.97,6.42	-
nersc	debug	4.95,21.12	1.32,0.94	-	-
nersc	low	2.78,2.93	fail	1.69,4.76	-
nersc	premium	1.90,F	1.76,1.70	-	-
nersc	regular	5.37,5.01	2.81,3.13	fail	-
nersc	reglong	1.53,3.77	-	-	-
sdsc	normal	5.39,F	2.08,7.64	1.74,3.36	-
sdsc	high	0.82,1.35	0.65,1.62	1.32,2.87	-
sdsc	low	1.36,2.54	1.27,2.42	1.96,9.73	-
sdsc	express	1.10,F	-	-	-
tacc2	normal	3.01,5.35	2.57,9.15	2.79,2.18	1.22,F
tacc2	devel	0.57,0.13	1.12,0.38	-	-
tacc2	serial	1.13,F	-	-	-

Table 2. Simulation results showing the mean square error of successful predictions for three prediction methods.

Table 1 shows the results of predicting the upper bound on the 0.95 quantile with 95% confidence for BMBP, Weibull, and log-normal in terms of success percentage. The first column shows the site and machine associated with each trace and shows the second column contains the queue name. The data in the remaining four columns shows which of the three methods (denoted *B, W, L* for Binomial, Weibull, log-normal respectively) achieves a success percentage of 95% or greater, rounded to the nearest percent. In cases where there is insufficient data, we show a dash, and the word *none* indicates that no method achieves 95%. Of the 55 traces with 1000 jobs or more, BMBP is successful for 51, the Weibull method succeeds in 52 cases, and the log-normal in 36 cases.

Note that while space considerations prevent a more thorough characterization of the data, both the number of jobs in each subdivision and the time period covered by that subdivision vary considerably. In general, the *normal* or *regular* queues see considerably more small job counts than the more "exotic" queue names and tend to span greater portions of each overall trace chronologically. As such, the time-series characteristics are quite different across the spectrum of these test cases.

Table 2 shows the ratio of root mean square over- prediction as defined in Section 5 of the Weibull and log-normal methods to that generated by BMBP. In each cell of the table, the first number is the ratio of Weibull to BMBP, and the second number is that for log-normal to BMBP. We indicate cases where either method failed to achieve 95% and BMBP is successful with the character *F*. Additionally, we denote cases where BMBP fails to achieve 95% with the word *fail*. We do not consider the error ratios in failed cases any prediction error is possible if the target success percentage is not achieved.

From Table 1, it is clear that the log-normal method is inferior to both BMBP and the Weibull method in terms of success rate: not only does it fail on the largest number of cases, but there are no cases where it is the only successful method. If we compare the more accurate BMBP and Weibull methods, we see from the ratios shown in Table 2 that BMBP is more accurate than the Weibull method. Among the 51 cases where both methods are successful, the Weibull method achieves a tighter bound only 7 times. The Weibull method does achieve a 95% success rate for a single case that fails for BMBP (NERSC, *low* queue, 5 - 16 processors) but as we discuss below, this failure would most probably be recored as a success if a few more jobs had been available in the trace. Moreover, the median root mean square over-prediction error ratio for Weibull to BMBP is 1.62. Thus, we assert that BMBP in aggregate yields tighter, hence more meaningful, bounds on queuing delay across the spectrum of test cases than does the Weibull, while achieving the same level of correctness on all but one of the cases tested.

6.2 Analysis of BMBP Failures

For four of the subdivided traces shown in Table 1 BMBP fail to achieve a success percentage of 95% or better. These failures occur as a result of two distinct conditions in the data. The first is best illustrated by the time-series representation of the queue waiting times and the corresponding predictions of them for the Datastar *high* queue for jobs requesting 1-4 processors, depicted in Figure 1.

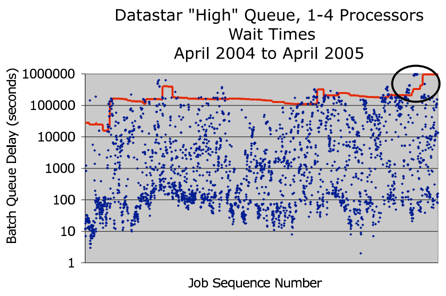

Figure 1. Measurements and Predictions For Datastar *high* Queue for 1-4 Processors Shown by Job Sequence Number.

Note that the circled region in the figure corresponds to a sequence of consecutive jobs that had unusually long wait times. These jobs were all in the queue *at the same time*, so that BMBP did not have the long wait time for the first job available in time to correct its predictions for the jobs that followed. This is an example of the correlated prediction errors in the discussion from Section 4, which are characteristic of this method. It also illustrates the point that BMBP is a long-run method; observe that for the last few values in the trace, the predictor detected a changepoint and adjusted its predictions accordingly. This type of behavior was observed in many other queues; however, in most of the traces, there were enough subsequent jobs that the overall success rate of BMBP climbed back above 95%. We also note that immediately before the long sequence of high values, the percentage of correct predictions was at better than 95% for the trace. The failure in the NERSC low queue for 5-16 processors was due to a similar phenomenon. A changepoint farther from the end of the trace (i.e. few more jobs submitted to the queue) would have most likely caused BMBP to record a success in this case.

The second type of failure is associated with the beginning of the trace and is attributable to "burn-in," as previously mentioned. More formally, we think of a "burn-in" phase as the period of time for which the data have not found any sort of limiting distribution. This may be due to heavy dependence on initial conditions, rapidly fluctuating policy, or other effects; in any event, it is a behavior characteristic of new machines and not observed in any trace after a sufficient amount of time has passed. Figure **??** depicts the queue wait times for the LANL small queue for jobs requesting 1-4 processors. Note that data within the circled regions in the first part of the trace exhibit strong upward trends – exactly the sort of data that BMBP cannot handle (again, see the discussion in Section 4). Indeed, if a series of wait-time data were characterized by trended periods, BMBP would fail to produce successful predictions. In this case, however, it seems to be due to the machine having been new at the time the trace started. We chose 30 days, somewhat arbitrarily, as a "burn-in" period for each trace, and for this particular machine, it is evident that the queue wait times did not reach any sort of "steady state" until later. We note that when we re-ran our predictor with the burn-in period to 60 days, our success rate is better than 95%. This phenomenon was also observed for the failure of BMBP in the NERSC regular queue for 17-64 processors.

Note that The total number of jobs contained in the four failed traces is $16,269$ which is approximately 1.4% of the total survey. Moreover, only a single trace generates a success percentage below 93% and in all failure cases BMBP is the most accurate (lowest error) method.

6.3 Characterizing Queue Delay for Users

The potential value of such predictions is illustrated in Figure 3. In the figure, we show the BMBP prediction of the upper bound on the 0.95 quantile with 95% confidence for February 24, 2005 in the *normal* queue on Datastar at SDSC and Lonestar at TACC. The black line shows the predicted queue delay for Datastar and the gray line, the delay for Lonestar. The units in the figure are seconds, and the y-axis is shown on a log scale.

From between approximately 6:50 AM and 3:25 PM on the 24th, a user with a choice between running a job (of any processor count) in the *normal* queue at SDSC and at TACC would have been able to predict that the job would have started in 12 seconds or less if submitted at TACC with at least 95% certainty. Similarly, the same user could have predicted that the job, if submitted at SDSC during the same 24-hour period, would have started execution in less than 4 days, with the same 95% certainty. We recognize that few users have the luxury, at present, of choosing between top-quality resources such as Lonestar and Datastar. However as grid computing [13, 2] becomes more prevalent, and multi-site resources such as TeraGrid [24] become more popular, we believe that the need for effective prediction of this type will be important.

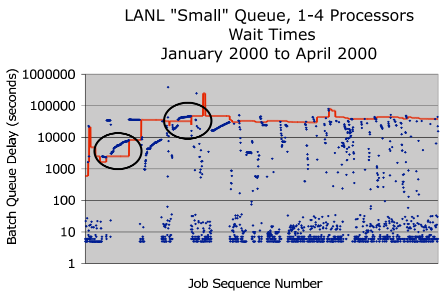

Figure 2. Measurements and Predictions For LANL *small* queue for 1-4 Processors Shown by Job Sequence Number.

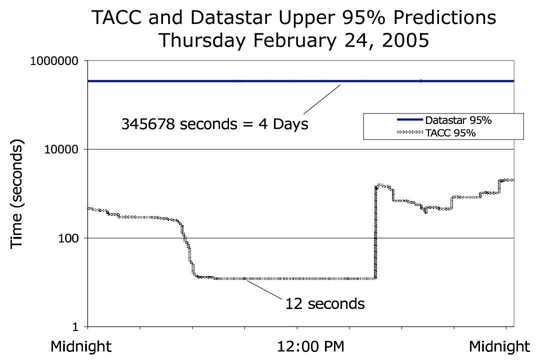

Figure 3. Predicted queue delay upper bounds on SDSC Datastar (black line) and 1 TACC Lonestar (gray line) for February 24, 2005

To attempt to realize this capability for HPC users, we have deployed BMBP at a variety of computing centers including SDSC and TACC (from whence we gathered data for this study) and the TeraGrid [24] sites. We have also implemented prototype web-based browsers for the data so that users can view predictions for the these systems generated by the Network Weather Service in real time. These browsers are currently accessible from `http://pompone.cs.ucsb.edu/~nurmi/batchq/nindex.html` and `http://pompone.cs.ucsb.edu/~rgarver/bqindex.php` in prototype form, however we plan to transition them into production the near future. Thus while we have demonstrated BMBP using an analysis of historical job data and a simple simulation, we note that it constitutes a functioning system with real deployments as well.

7. Conclusion

High-performance computing centers rely heavily on space-sharing systems to support their users' computational demands. These systems typically employ a batch scheduler to handle multiple jobs requesting access to the machines, which introduces queuing delays that users experiences as delay in job turn-around time. While users can usually predict job execution time once scheduled, queuing delay, which can often exceed execution time, is more difficult to predict.

In this work, we propose a novel batch job wait time prediction method which uses as input a historical trace of job wait times, a quantile of interest (corresponding to a level of certainty as to how soon the job will execute), and a confidence bound on the quantile prediction. With this information, the BMBP method can produce a prediction for the specified quantile at the given confidence level which we have shown to be both reliable and robust in simulation. Our experiment compares the BMBP method to model-fitting methods based on the Weibull and log-normal distributions and finds it superior to both. In particular, it is more correct than the method that uses log-normal distributions and more accurate than the one that uses Weibull distributions.

8. REFERENCES

[1] IBM LoadLeveler User's Guide. Technical report, International Business Machines Corporation, 1993.

[2] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.

[3] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis, Forecasting, and Control, 3rd edition*. Prentice Hall, 1994.

[4] J. Brevik, D. Nurmi, and R. Wolski. Quantifying machine availability in networked and desktop grid systems. In *Proceedings of CCGrid04*, April 2004.

[5] S.-H. Chiang and M. K. Vernon. *Dynamic vs. Static Quantum-based Processor Allocation*. Springer-Verlag, 1996.

[6] S. Clearwater and S. Kleban. Heavy-tailed distributions in supercomputer jobs. Technical Report SAND2002-2378C, Sandia National Labs, 2002.

[7] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[8] A. Downey. Using queue time predictions for processor allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.

[9] The Dror Feitelson's Parallel Workload Page. `http://www.cs.huji.ac.il/labs/parallel/workload`.

[10] D. G. Feitelson and B. Nitzberg. *Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*. Springer-Verlag, 1996.

[11] D. G. Feitelson and L. Rudolph. *Parallel Job Scheduling: Issues and Approaches*. Springer-Verlag, 1995.

[12] D. G. Feitelson and L. Rudolph. *Towards Convergence in Job Schedulers for Parallel Supercomputers*. Springer-Verlag, 1996.

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

[14] E. Frachtenberg, D. G. Feitelson, J. Fernandez, and F. Petrini. *Parallel Job Scheduling Under Dynamic Workloads*. Springer-Verlag, 2003.

[15] C. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.

[16] Gridengine home page – `http://gridengine.sunsource.net/`.

[17] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, June 1999.

[18] D. Lifka. *The ANL/IBM SP scheduling system*, volume 949. Springer-Verlag, 1995.

[19] D. Lifka, M. Henderson, and K. Rayl. Users guide to the argonne SP scheduling system. Technical Report TM-201, Argonne National Laboratory, Mathematics and Computer Science Division, May 1995.

[20] B. Lindgren. *Statistical Theory*. MacMillan, 3 edition, 1968.

[21] Maui scheduler home page – `http://www.clusterresources.com/products/maui/`.

[22] D. Moore. *The Basic Practice of Statistics*. W.H. Freeman, 2 edition, 2000.

[23] Cray NQE User's Guide – `http://docs.cray.com/books/2148_3.3/html-2148_3.3`.

[24] NSF TeraGrid Project. `http://www.teragrid.org/`.

[25] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of Europar 2005*, August 2005.

[26] D. Nurmi, R. Wolski, and J. Brevik. Model-based checkpoint scheduling for volatile resource environments. In *Proceedings of Cluster 2005*, September 2004.

[27] The network weather service home page – `http://nws.cs.ucsb.edu`.

[28] Pbspro home page – `http://www.altair.com/software/pbspro.htm`.

[29] W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 202–219, London, UK, 1999. Springer-Verlag.

[30] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999.