Predicting Bounds on Queuing Delay in Space-shared Computing Environments
University of California, Santa Barbara Technical Report Number CS2005-09

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

Abstract

Most space-sharing resources presently operated by high performance computing centers employ some sort of batch queueing system to manage resource allocation to multiple users. In this work, we explore a new method for providing end-users with predictions of the bounds on queuing delay individual jobs will experience when waiting to be scheduled to a machine partition. We evaluate this method using scheduler logs that cover a 9 year period from 7 large HPC centers. Our results show that it is possible to predict delay bounds with specified confidence levels for jobs in different queues, and for jobs requesting different ranges of processor counts.

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.

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 "job" [1] along with a description of any parameters and environmental inputs (*e.g.* input files, shell environment variables, *etc.*) 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 [20], LoadLeveler [1], EASY [16], NQS/NQE [18], Maui [17] and GridEngine [12] 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 (in order to ensure 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 *Brevik 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 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

[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.

in this study (7 supercomputers operated by the National Science Foundation and the Department of Energy over a 9-year period) is highly . 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 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 potentially changing under administrator control. Thus, while the implementation of multiple policies for a given machine through multiple queues can provide a high level and qualitative expectation of how a specific job will be treated, it substantially complicates the problem of making a quantitative prediction for that job.

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 [15] 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 [21, 5, 6, 8, 11, 3, 7, 9] 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.

Our work differs from these approaches in two significant ways. First, 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. We focus only on the problem of prediction at the expense of a complete statistical model of system behavior. As a result, BMBP achieves new levels of predictive accuracy and quantitative rigor, but it cannot easily be used to build simulations of future or hypothetical systems in the same way previous results can.

Second, BMBP makes a prediction for each individual job's queuing delay rather than a statistical characterization of the queuing delay experienced by all jobs. For example, previous efforts have focused on describing job behavior using different parametric models so that the mean queuing delay can be estimated. It is difficult to quantify and predict how the delay that will be experienced by a job that is about to be submitted will compare to the estimated mean delay. In contrast, BMBP correctly predicts bounds on delay for individual jobs (rather than the collection of all jobs) with quantifiable confidence levels.

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 log-normal. Our results show that our new approach achieves the specified confidence levels in each case while the log-normal approach does not.

We emphasize that our intention in developing BMBP is to provide a practically realizable predictive capability for eventual deployment as a user and scheduling tool rather than a new analytical methodology. Therefore our reportage focuses on the results generated by a work prototype that is currently being integrated with various batch scheduling systems, and our results are, ultimately, empirical.

2. Related Work

Smith, Taylor, and Foster in [21] 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 [14, 4] 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 with 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 [5, 6] uses a 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 odds that their job will fall outside the range.

3. Problem Definition: Predicting Bounds on Queuing Delay

While it is appealing to consider the problem of predicting the exact queuing delay a given job will experience through some deterministic mechanism, because the scheduling policy is hidden and potentially changing, and because there is no deterministic model for job interarrival duration or execution time, we assert that queuing delay must be treated statistically as well. As a result, 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.

For example, 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.

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 (perhaps somewhat unfortunately) 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 direct 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 the upper bound on queuing delay, it can be similarly formulated in terms of produce lower confidence bounds, or two-sided confidence intervals, at any desired level of confidence, for any population quantile.

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 of the queuing delay, and to report the degree of conservatism as the quantified confidence level.

4.1 The Brevik Method Batch Predictor

Our approach, which we term the *Brevik 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 $(1-q)$. Thus (under suitable assumptions about 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 k and n. Therefore, the probability that k or fewer observations are greater than X_q is equal to

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

We provide a more complete description of the method in the Appendix of this paper. However, in short, we can find the smallest value of k for which Equation 1 is larger than some specified confidence level, and the k^{th} value in a sorted set of observations (of sufficient size) will be greater than or equal to the X_q quantile of the distribution from which the observations were made with the specified level of confidence.

Nonstationarity

In the specific context of batch-queue wait times, one certainly cannot make the unsupported assumption that the data are i.i.d. (independent and identically distributed); therefore, the idea mentioned above that the wait time for a *specific job* is to be regarded as a random variable with its own specific distribution returns to the foreground. In fact, we have observed that, over the course of a long data trace, the above prediction method, which essentially regards a specific wait time as simply a particular instance of a general "wait time" random variable, is subject to degradation. We assume that this degradation is because the system is changing over time making the observation sequence nonstationary.

We circumvent this difficulty in the following way. First, observe that, given an i.i.d. sequence of data from a random variable X and a sequence value x_i that is greater than, say, $X_{.95}$, the probability that the next value x_{i+1} is also greater than $X_{.95}$ is .05, which is low but not exceptional; however, the probability that the next two are greater than $X_{.95}$ is $.0025$ – an extremely rare occurrence. Therefore, if we are trying to make inferences about the $.95$ quantile of a data set and we find three measurements in a row that exceed their .95-quantile estimates, we can be almost certain that this has taken place due to some nonstationarity in the data rather than purely by chance from a stationary sequence.

Now, suppose that the data, regarded as a time series, exhibits some autocorrelation structure. If the first autocorrelation is fairly strong, three or even five measurements in a row above the $.95$ quantile might not be such a rare occurrence, since one high value would tend to produce another. For this reason, we conducted a Monte Carlo simulation using log-normal distributions with various values of first autocorrelation in order to identify the number of consecutive measurements above the $.95$ quantile necessary to constitute a "rare event," meaning one that occurs for less than 5% of the errors. We then generated a coarse-grained lookup table with autocorrelations and "rare-event" thresholds. Note that our choice of log-normal distribution does not amount to *assuming* that the distributions of wait times are log-normal; we only use this distribution to give a very rough sense of how many consecutive incorrect predictions constitute a "rare event," as characterized above, for heavy-tailed autocorrelated data. As a practical matter, the log-normal is an easy distribution to work with, and in in light of Downey's below-mentioned endorsement of log-normals, as well as the fact that log-normals did in fact perform fairly well as prediction models, log-normal distributions seem to be a good choice for our simulation. In any event, it should be noted that it is not vital to our method that the log-normal be entirely accurate: Our choice of 5% was somewhat arbitrary to begin with, and if the number produced by this method turns out to give an event that occurs with 3% or 7% probability, it can hardly be construed as a serious difficulty.

For each queue, we calculate the first autocorrelation during the training period and use the lookup table to find the "rare-event" threshold for that data set. When we observe the determined number of consecutive incorrect predictions, 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 $j = n - 1$, so that the sum gives the probability that $n-1$ or fewer are less 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 and start making predictions based on the shortened history. Thus our confidence bounds automatically adapt 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 were correct at least 95% of the time. The relatively high level of confidence chosen enabled the predictor to work well in spite of possible effects of non-independence and short-term nonstationarity in the data.

4.2 Model-Fitting with Log-Normals

In [5], Downey hypothesizes that the job at the head of a FCFS queue experiences a delay that is well-modeled by a *log-uniform* distribution; however, in a private communication with the author,

he expressed a belief that overall wait times are well modelled by *log-normal* distributions; note that that a random variable X is distributed log-normally if $\log X$ is a normally-distributed variable. This observation suggests another approach to the problem of producing quantile estimates for batch-queue wait times; specifically, one can fit a log-normal to the data (or, equivalently, a normal distribution to their logarithms) using, preferably, the method of maximum likelihood estimation (MLE), and then produce the desired population quantile from a lookup table or the inverse of the cumulative distribution function.

In fact, the above method will likely produce accurate quantile estimates for a true log-normal; however, in the interest of statistical rigor, as well as an "apples-to-apples" comparison with BMBP, we produce confidence upper bounds for quantiles rather than quantile estimates themselves. To this end, we work with the logarithms of the data; since they are assumed to be normal, we can use the K' distribution for confidence bounds on quantiles for normal populations, given in Table 4.6 of [13].

Generally, model-fitting is done with all of the data available, and our experiments include the use of full histories to produce confidence bounds; however, in light of the long-term nonstationarity phenomenon discussed above, we additionally implemented an estimation scheme incorporating the same history-truncation strategy that we used with BMBP. The result separates the effects of using a binomial approach from the effects of our automatic identification of change points.

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 level of confidence. We examine several different combinations of quantile and confidence level as part of this verification. As a measure of accuracy, we detail the degree of over-prediction each upper bound generates. 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 plan to deploy BMBP in production computing settings, 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 an alternative approach based on a dynamically fit log-normal distribution determined by an MLE (as described previously) 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 either the BMBP or log-normal prediction system 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 believe that the correct and accurate behavior of BMBP in this setting, however, warrants deployment and "live" evaluation as a next step.

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 1.2 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 three things happen.

- The virtual time specified for the job to wait in the pending expires.
- A new job enters the system according to the virtual clock.
- A specified number of seconds elapses (specified as an input parameter) allowing the prediction method to "refit" its models.

When the first case occurs, the job is simply added to a growing list of historical job wait times stored in memory. Although the waiting time for the job is carried in the trace, the predictor is not entitled to "see" the waiting time in the history until it stops waiting in queue and is released for execution.

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 could only be determined after the job completed its waiting period.

When the third case occurs, the simulation makes a new prediction based on the current contents of the historical data buffer. This prediction is used for all jobs until another time epoch has elapsed. The reason why we wait a set number of seconds before making new predictions (case 3) instead of making new predictions every time a new job enters the queue (case 2) is to simulate

a real world problem; we do not expect to actually have real-time access to job wait time data. Instead, we assume that the predictor will get an up-to-date "dump" periodically (every five minutes for the results reported in the next section). We note that we have run simulations for which the epoch length is set to 0 seconds, simulating the (likely unrealizable) deployment scenario in which the predictor state is updated for each job, and the effect on the results was minimal.

Also, because predictions are based on history, we train each simulation using an initial fraction (10% in this study) of each job sequence. During the training period, the simulation executes as above but does not record the successes or failures of the predictions begin made; in effect, it is simply putting the specified fraction of job wait times in a historical buffer.

When the training period is exhausted, the simulation enters the result phase. In this phase, the code executes as described, making quantile predictions with a given confidence bound, and recording a success or failure for each job entering the queue. The simulator also records the ratio of the prediction to the observed wait time for each job. We use the median of these ratios to measure the accuracy of each simulation run (as described in Section 6).

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.

In this work, we consider only the historical job submission information associated with each queue and machine. While in some cases the queue names suggest a potential prioritization (as with the NERSC queues), the logs do not include the priority relationship between queues explicitly. Furthermore, because administrators may change the prioritization in a way that is not made completely explicit, we believe that prediction technology based strictly on observed queuing behavior is desirable.

Abbreviation	Detail
Datastar	Power4 P690 parallel computer built by the IBM Corporation
Cray-Dell	Dell x86 processors, built by Cray Research Incorporated
LANL	Los Alamos National Laboratory located in Los Alamos, New Mexico and operated for the Department of Energy
NERSC	National Energy Research Center located at the University of California, Berkeley and operated or the Department of Energy
O2K	Origin 2000 parallel computer built by Silicon Graphics Corporation
Paragon	Paragon parallel computer built the Intel Corporation
SDSC	San Diego Supercomputer Center located at the University of California, San Diego and operated for the National Science Foundation
SP	SP-series parallel computer built by the IBM Corporation
TACC	Texas Advanced Computing Center located at the University of Texas, Austin and operated for the National Science Foundation

Table 2. Abbreviations used in Table 1

Table 1 summarizes the log traces we consider in this study. Column 1 of the table shows the site and machine type of each trace in abbreviated form. We detail the abbreviations in Table 2; we note that each site is a production computing facility with a predominantly scientific computing user community operated either for the U.S. National Science Foundation or the Department of Energy. In the second column of Table 1 we show the period of time covered by each trace (in terms of month and year). Column 3 shows the alphanumeric queue name listed in each log and column 4 indicates the number of records in each trace. In Columns 5 and 6 we show, respectively, the mean and median queuing delays in seconds. Finally, column 7 depicts the sample standard deviation from each trace.

From the last three columns of the table, it is clear that the distribution of queue delays for each queue on each machine at each site is heavy-tailed: In each case, the median wait time is significantly less than the average wait time, and the variance is large relative to the average. A large body of previous work [5, 21, 8, 14, 4] has observed that the interarrival and job execution times for parallel jobs are also typically heavy-tailed.

This collection of data comprises 1.26 million jobs covering 9 years at the San Diego Supercomputer Center, Lawrence Berkeley Laboratory, Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and the Texas Advanced Computing Center. It includes a variety of job mixes, machine types, and priority schemes (some of which are unknown), and thus it represents a substantial "sample" of batch-queue user experience in terms of observed queue delay.

6. Results

Site/Machine	Date	Queue Name	Job Count	Avg. Delay	Median Delay	Std. Deviation
SDSC/Datastar	4/04 - 4/05	TGhigh	1488	29589	6269	64832
SDSC/Datastar	4/04 - 4/05	TGnormal	5445	7333	88	28348
SDSC/Datastar	4/04 - 4/05	express	11816	2585	153	11286
SDSC/Datastar	4/04 - 4/05	high	5176	35609	1785	100817
SDSC/Datastar	4/04 - 4/05	high32	606	13407	251	32313
SDSC/Datastar	4/04 - 4/05	interactive	5822	1117	1	10389
SDSC/Datastar	4/04 - 4/05	normal	48543	35886	1795	100255
SDSC/Datastar	4/04 - 4/05	normal32	5322	24746	1234	61426
SDSC/Datastar	4/04 - 4/05	normalL	727	48432	1337	97090
LANL/O2K	12/99 - 4/00	chammpq	8102	6156	33	13926
LANL/O2K	12/99 - 4/00	irshared	1012	1779	6	17063
LANL/O2K	12/99 - 4/00	medium	880	11570	1670	21293
LANL/O2K	12/99 - 4/00	mediumd	1552	1448	296	8039
LANL/O2K	12/99 - 4/00	scavenger	50387	1433	7	7126
LANL/O2K	12/99 - 4/00	schammpq	1386	7955	8450	8481
LANL/O2K	12/99 - 4/00	shared	35510	1094	6	6752
LANL/O2K	12/99 - 4/00	short	2639	4417	13	11611
LANL/O2K	12/99 - 4/00	small	14544	22098	67	81742
LLNL/Blue Pacific	1/02 - 10/02	all	63959	8164	242	18245
NERSC/SP	3/01 - 3/03	debug	115105	332	42	3950
NERSC/SP	3/01 - 3/03	interactive	36672	121	1	2417
NERSC/SP	3/01 - 3/03	low	56337	34314	6020	91886
NERSC/SP	3/01 - 3/03	premium	24318	3987	177	15103
NERSC/SP	3/01 - 3/03	regular	274546	16253	1578	47920
NERSC/SP	3/01 - 3/03	regularlong	3386	57645	43237	64471
SDSC/Paragon	1/95 - 1/96	q1l	5755	16319	10205	27086
SDSC/Paragon	1/95 - 1/96	q256s	1076	808	7	7477
SDSC/Paragon	1/95 - 1/96	q32l	1013	4301	8	12565
SDSC/Paragon	1/95 - 1/96	q64l	3425	4324	11	11240
SDSC/Paragon	1/95 - 1/96	standby	8896	14602	604	35805
SDSC/SP	4/98 - 4/00	express	4978	1135	22	4224
SDSC/SP	4/98 - 4/00	high	8809	16545	567	133046
SDSC/SP	4/98 - 4/00	low	22709	20962	34	95107
SDSC/SP	4/98 - 4/00	normal	30831	26324	89	101900
TACC/Cray-Dell	1/04 - 3/05	development	5829	74	9	1850
TACC/Cray-Dell	2/04 - 12/04	hero	48	28636	12	71168
TACC/Cray-Dell	2/04 - 3/05	high	2110	5392	10	33366
TACC/Cray-Dell	1/04 - 3/05	normal	356487	732	10	9436
TACC/Cray-Dell	8/04 - 3/05	serial	7860	2178	10	13702

Table 1. Job submittal traces. The units for the mean, median and standard deviation measurements are seconds.

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 nonstationary nature of each series – there is considerable variation among the various methodologies we tested in terms of their accuracy and computational cost.

6.1 Predicting By Queue Name

We begin by comparing the efficacy of using BMBP for quantile prediction in batch queues to that of parametric inference based on the assumption that the data are distributed log-normally, both with and without the history-trimming features discussed above. In this first set of simulation experiments, the data used for these simulations is subdivided according to the queue name recorded in each log file but is not further subdivided by processor count. That is, in each case, we predict the bounds on the queuing delay for a job submitted to a particular queue without regard to its requested number of processors. We report the observed fraction of successful predictions for each queue when we predict the 0.95 quantile with an upper confidence level of 95%. The results for simulations covering all of the logging data are shown in Table 3. The first column indicates the site, or in the cases where we have investigated multiple machines at a site, the machine name. The second column indicates the name of the queue. In columns 3, 4, and 5 we show the prediction results for BMBP, the log-normal method without history trimming, and the log-normal method with the BMBP history trimming technique. Any value that falls below 0.95 (indicating that the method has failed to provide upper bounds for the 0.95 quantile for that queue) is marked with an asterisk. In addition, for each of the three prediction methods, we boldface the technique for which the median prediction ratio (described in Subsection 5.1 and shown in Table 4) is the lowest. As Table 3 indicates, BMBP correctly predicts an upper bound for at least 95% of the jobs submitted to each queue except the *short* queue at LANL. In this particular queue, 8% of the jobs occur at the very end of the log with unusually long delays. It may be that if more of the log were available, the eventual correct prediction fraction would be above 0.95 or it may be that the method is simply unable to adapt quickly enough to the sudden changes in delay. For all of the other queues, the fraction is above 0.95 and for a large majority of the queues, BMBP gives the tightest upper bound (indicated by the boldfaced values). Using a log-normal model (even one that is refit every 5 minutes) that considers all previously observed measurements in each prediction works well for some queues but fails for others. Therefore, using full histories to model the population of queue delays with a log-normal model fails as a general methodology. To test the the robustness of the binomial approach used in BMBP against that of the method based on assuming log-normal distributions, we show the performance of the log-normal using the same history trimming scheme employed by BMBP. While our history trimming method improves the performance of the log-normal method (both in terms of correctness and accuracy), there are still queues for which BMBP is the only general method of the three which is able to produce correct predictions for the 0.95 quantile for each job in each queue on each machine.

Machine	Queue	BMBP	logn NoTrim	logn Trim
datastar	TGhigh	**0.95**	0.92*	0.96
datastar	TGnormal	0.98	0.91*	**0.95**
datastar	express	**0.98**	0.92*	0.94*
datastar	high	**0.97**	0.91*	0.97
datastar	normal	**0.95**	0.93*	0.96
datastar	normal32	**0.97**	0.90*	0.98
lanl	schammpq	**0.97**	1.00	1.00
lanl	mediumd	**0.97**	0.97	0.97
lanl	short	0.91*	0.86*	0.87*
lanl	chammpq	0.97	**0.98**	0.98
lanl	small	**0.97**	0.98	0.98
lanl	shared	**0.97**	0.89*	0.93*
lanl	scavenger	0.97	0.92*	**0.96**
llnl	all	**0.97**	1.00	1.00
nersc	debug	0.97	0.95	**0.95**
nersc	interactive	0.97	0.87*	**0.95**
nersc	low	**0.96**	0.99	0.99
nersc	premium	0.97	0.96	**0.96**
nersc	regular	**0.97**	0.98	0.98
nersc	regularlong	**0.97**	1.00	1.00
paragon	q256s	0.97	0.95	**0.95**
paragon	q64l	0.98	**0.98**	0.99
paragon	q1l	**0.97**	1.00	1.00
paragon	standby	**0.98**	0.99	0.98
sdsc	normal	**0.97**	0.93*	0.98
sdsc	high	0.96	**0.95**	0.98
sdsc	low	0.97	0.90*	**0.98**
sdsc	express	**0.97**	0.84*	0.94*
tacc2	normal	0.99	0.96	**0.98**
tacc2	high	0.99	**0.97**	0.97
tacc2	development	0.98	**0.97**	0.98
tacc2	serial	0.97	0.89*	**0.96**

Table 3. Simulation results indicating percentage of correct job wait time predictions for all jobs in each queue.

The potential value of such predictions is illustrated in Figure 1. 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 recog-

Machine	Queue	BMBP	logn NoTrim	logn Trim
datastar	TGhigh	**4.55e-02**	6.39e-02*	1.92e-02
datastar	TGnormal	2.18e-03	9.16e-03*	**6.63e-02**
datastar	express	**1.02e-02**	2.89e-02*	2.85e-02*
datastar	high	**9.88e-03**	1.92e-02*	7.12e-03
datastar	normal	**9.43e-03**	1.11e-02*	7.78e-03
datastar	normal32	**1.80e-02**	3.21e-02*	1.05e-02
lanl	schammpq	**3.93e-01**	4.51e-02	4.69e-02
lanl	mediumd	**3.56e-02**	3.33e-02	3.19e-02
lanl	short	5.90e-04*	2.34e-03*	1.37e-03*
lanl	chammpq	9.22e-04	**1.01e-03**	6.80e-04
lanl	small	**4.59e-04**	3.26e-04	1.86e-04
lanl	shared	**1.25e-03**	1.07e-02*	2.02e-02*
lanl	scavenger	1.35e-03	3.15e-03*	**5.58e-03**
llnl	all	**4.24e-03**	1.27e-03	1.27e-03
nersc	debug	3.48e-02	5.47e-02	**6.07e-02**
nersc	interactive	1.08e-02	6.48e-02*	**3.03e-02**
nersc	low	**1.37e-02**	6.73e-03	4.62e-03
nersc	premium	6.81e-03	8.74e-03	**1.13e-02**
nersc	regular	**1.39e-02**	8.46e-03	8.75e-03
nersc	regularlong	**2.19e-01**	5.64e-02	5.64e-02
paragon	q256s	1.29e-03	4.41e-03	**8.16e-03**
paragon	q64l	2.95e-04	**3.38e-04**	3.04e-04
paragon	q1l	**9.60e-02**	5.93e-02	4.21e-02
paragon	standby	**3.48e-03**	2.15e-03	2.39e-03
sdsc	normal	**7.93e-04**	1.20e-03*	5.76e-04
sdsc	high	9.05e-03	**1.09e-02**	5.98e-03
sdsc	low	4.08e-03	1.92e-03*	**4.20e-03**
sdsc	express	**2.38e-03**	1.72e-02*	8.44e-03*
tacc2	normal	4.88e-03	2.78e-02	**2.92e-02**
tacc2	high	2.38e-04	**1.19e-03**	1.10e-03
tacc2	development	3.75e-01	**3.81e-01**	3.20e-01
tacc2	serial	2.18e-03	2.10e-02*	**1.90e-02**

Table 4. Simulation results reporting median ratio of actual wait times over predicted wait times for three prediction methods.

nize that few users have the luxury, at present, of choosing between top-quality resources such as Lonestar and Datastar. However as grid computing [10, 2] becomes more prevalent, and multi-site resources such as TeraGrid [19] become more popular, we believe that the need for effective prediction of this type will be important.

6.2 Predicting By Queue Name and Processor Count

With scheduling improvements such as backfilling [15] and dynamically changing user priorities (often at the behest of besieged system administrators or center personnel struggling to meet the requirements of an important demonstration), users of modern batch systems have come to expect that processor count affects queuing delay. In particular, it is generally but somewhat anecdotally believed that "smaller" jobs (in terms of requested processor count) are given qualitatively higher priority than larger jobs, based on the assumption that it is easier to find "space" for smaller jobs. Thus, a common user desire is to be able to predict, at any point

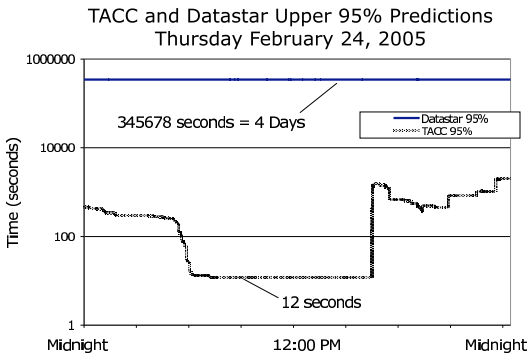

Figure 1. Predicted queue delay upper bounds on SDSC Datastar (black line) and 1 TACC Lonestar (gray line) for February 24, 2005

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. The specific range values (as shown in the top row of Table 5) were suggested by Karl Schulz and Jay Boisseau of the Texas Advanced Computing Center (TACC) as being the ones most meaningful to their user community.

Tables 5, 6, and 7 again show the results of predicting the upper bound on the 0.95 quantile with 95% confidence for BMBP, log-normal without history trimming, and log-normal with history trimming respectively. As before, boldface values indicate that the method which produced the given result not only was successful in terms of correct predictions, but is also the most accurate of the three methods, and asterisked values highlight the failure to achieve the desired 0.95 fraction of correct predictions. Also, 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. We denote these cases with a "-" in each table.

When job queues are broken down according to processor count, BMBP is clearly the most effective approach. The method is, again, generally correct as evidenced by the absence of asterisked values in Table 5: BMBP makes the desired percentage of correct predictions in each case. In contrast, the log-normal approach, either with or without history trimming, succeeds in some cases but fails in others. In addition, BMBP is the most accurate of the three approaches we have tested, as can be seen from the count of the boldfaced values in each table.

We illustrate the utility of these types of predictions in Figure 2. In the figure, we show the upper 0.95 quantile predictions with 95% confidence generated by BMBP for Datastar at SDSC during

Machine	Queue	1-4	5-16	17-64	65+
datastar	TGhigh	**0.95**	-	-	-
datastar	TGnormal	0.98	-	-	-
datastar	express	**0.98**	0.96	-	-
datastar	high	**0.95**	0.97	-	-
datastar	normal	**0.97**	0.97	0.96	-
datastar	normal32	**0.97**	-	-	-
lanl	schammpq	-	-	**0.98**	-
lanl	mediumd	-	-	-	**0.97**
lanl	short	-	-	**0.97**	-
lanl	chammpq	0.96	**0.96**	**0.97**	-
lanl	small	**0.96**	**0.95**	**0.98**	0.98
lanl	shared	**0.97**	**0.97**	-	-
lanl	scavenger	**0.97**	**0.98**	0.97	0.98
llnl	all	0.97	**0.98**	**0.98**	-
nersc	debug	0.97	0.97	-	-
nersc	interactive	0.97	-	-	-
nersc	low	**0.97**	**0.97**	**0.96**	-
nersc	premium	0.97	0.98	-	-
nersc	regular	**0.97**	**0.97**	**0.97**	-
nersc	regularlong	**0.96**	-	-	-
sdsc	normal	**0.96**	**0.96**	**0.97**	-
sdsc	high	**0.97**	**0.95**	**0.96**	-
sdsc	low	0.95	**0.95**	**0.96**	-
sdsc	express	**0.97**	-	-	-
tacc2	normal	**0.98**	0.98	**0.98**	0.98
tacc2	development	**0.98**	0.98	-	-
tacc2	serial	0.97	-	-	-

Table 5. BMBP simulation results indicating percentage of correct job wait time predictions.

Machine	Queue	1-4	5-16	17-64	65+
datastar	TGhigh	0.92*	-	-	-
datastar	TGnormal	0.91*	-	-	-
datastar	express	0.92*	0.91*	-	-
datastar	high	0.86*	**0.96**	-	-
datastar	normal	0.92*	**0.95**	**0.96**	-
datastar	normal32	0.90*	-	-	-
lanl	schammpq	-	-	1.00	-
lanl	mediumd	-	-	-	0.97
lanl	short	-	-	0.93*	-
lanl	chammpq	**0.96**	0.85*	1.00	-
lanl	small	1.00	1.00	1.00	0.87
lanl	shared	0.90*	-	-	-
lanl	scavenger	0.98	0.94*	0.95	0.87*
llnl	all	0.96	1.00	1.00	-
nersc	debug	0.95	0.98	-	-
nersc	interactive	0.87*	-	-	-
nersc	low	0.98	0.99	0.99	-
nersc	premium	0.94*	**0.95**	-	-
nersc	regular	0.98	0.98	1.00	-
nersc	regularlong	1.00	-	-	-
sdsc	normal	0.86*	0.99	1.00	-
sdsc	high	0.88*	0.98	1.00	-
sdsc	low	0.97	0.99	1.00	-
sdsc	express	0.86*	-	-	-
tacc2	normal	0.95	0.96	0.92*	0.93*
tacc2	development	0.98	**0.96**	-	-
tacc2	serial	0.89*	-	-	-

Table 6. Log-normal without history-trimming method simulation results indicating percentage of correct job wait time predictions.

the month of June, 2004. The black line indicates the number of seconds predicted for jobs requesting between 1 and 4 processors, and the gray line shows the predicted time for jobs requesting 17 to 64 processors. A user, furnished with these predictions, would have been able to correctly predict that the worst-case wait time for larger jobs would be *lower* than for smaller jobs. We found this result so surprising that we investigated the logs in detail and discovered that, in fact, larger jobs were favored for this month in terms of queuing delay (we omit this more detailed verification due to space constraints). Thus BMBP, had it been available, would have been able to forecast correctly the advantage of submitting larger jobs to the interested user.

6.3 Characterizing Queue Delay

The results thus far reported show that, in general, the BMBP method used with a confidence level of 95% provides fast, correct, and accurate upper bounds for the 0.95 quantile. As mentioned previously, the method can predict both upper and lower bounds for any quantile at a specified confidence level. To further illustrate the potential utility of such predictions, we show quantile bounds for the "Normal" queue serving the Datastar machine at SDSC on May 5th, 2004. Table 6.3 shows the lower bound on the 0.25 quantile and the upper bounds on the 0.5, 0.75, and 0.95 quantiles at 95% confidence generated every two hours from the logs.

Until 2:00 PM, the predicted quantiles all indicate the probability of a potentially long (greater than 4-hour long) queuing delay is at least 50%. Later in the day, however, the predicted bounds improve substantially, to the point where the predicted upper bound on the 0.5 quantile is approximately 24 minutes, and there is at least a 75% chance that a job will wait no more than 6.5 hours (just before midnight). Given these types of predictions for all of the queues at a site and/or multiple sites (both with and without categorization by processor count), and the assurance that they are correct to a specific level of confidence, we believe BMBP will provide an important new capability to HPC users.

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 leads to a problem of imposing batch job queue delays on user jobs. While users can reliably predict how long their job will take to execute once scheduled, they have not previously been able to predict how long their job will stay in the job queue. In this work, we propose a novel batch job wait time prediction method which uses as input a historical trace of job wait times, and quantile of interest, and a

Machine	Queue	1-4	5-16	17-64	65+
datastar	TGhigh	0.96	-	-	-
datastar	TGnormal	**0.95**	-	-	-
datastar	express	0.93*	**0.96**	-	-
datastar	high	0.97	0.99	-	-
datastar	normal	0.96	0.97	0.99	-
datastar	normal32	0.98	-	-	-
lanl	schammpq	-	-	1.00	-
lanl	mediumd	-	-	-	0.97
lanl	short	-	-	0.94*	-
lanl	chammpq	0.96	0.92*	1.00	-
lanl	small	1.00	1.00	1.00	**0.97**
lanl	shared	0.93*	0.97	-	-
lanl	scavenger	0.97	0.94*	**0.96**	**0.97**
llnl	all	**0.97**	1.00	1.00	-
nersc	debug	**0.96**	**0.97**	-	-
nersc	interactive	**0.95**	-	-	-
nersc	low	0.99	1.00	1.00	-
nersc	premium	**0.96**	0.98	-	-
nersc	regular	0.97	0.99	1.00	-
nersc	regularlong	1.00	-	-	-
sdsc	normal	0.93*	0.99	1.00	-
sdsc	high	0.93*	0.99	1.00	-
sdsc	low	**0.96**	0.99	1.00	-
sdsc	express	0.93*	-	-	-
tacc2	normal	0.98	**0.97**	0.97	**0.95**
tacc2	development	0.99	0.97	-	-
tacc2	serial	**0.96**	-	-	-

Table 7. Log-normal with trimming method simulation results indicating percentage of correct job wait time predictions.

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 compared the BMBP method, a more traditional log-normal method, and a log-normal method with a history trimming technique employed by the BMBP. The BMBP method was more correct and accurate in general than either log-normal method both in the case where job wait time data was subdivided by node count ranges and when all job sizes were considered.

Appendix: BMBP Details

Recall Formula 1 from Section 4, which established, for an i.i.d. sample (x_i) from a random variable X, that the probability that k or fewer of the x_i are greater than X_q is equal to

$$\sum_{j=0}^{k} \binom{n}{j} \cdot (1-q)^{n-j} \cdot q^j \qquad (2)$$

Observe that this calculation is valid (not just asymptotically correct) under the sole assumption that the x_i are independent and identically distributed (i.i.d.), and also that it depends only on n, k, and q.

.25 Quantile	.5 Quantile	.75 Quantile	.95 Quantile
44	4769	41058	159844
143653	301970	343518	471515
140115	301970	343518	471515
140115	301970	343518	471515
28	227527	343518	471515
44	299676	388933	521723
42	15788	301970	455116
22	11323	197811	400614
103	4053	168348	400614
85	3165	143790	400610
134	1944	23836	396776
118	1940	23787	396776
102	1465	23606	396776

Table 8. One day in the life of the datastar/normal queue showing prediction quartiles of interest.

Given a desired confidence level C and quantile of interest X_q, we can use Equation 2 above to obtain a level-C upper bound for X_q. Let $x_{(i)}, i = 1, \ldots, n$, represent the *order statistics*; that is, $(x_{(1)}, x_{(2)}, ..., x_{(n)})$ permutes the sample so that it is in increasing order. To say that we are confident with level C that $x_{(k)} > X_q$ is equivalent to saying that the *a priori* probability that $x_{(k)} > X_q$ is greater than or equal to C; by Equation 2, this gives the equation

$$\sum_{j=0}^{k} \binom{n}{j} \cdot (1-q)^{n-j} \cdot q^j \geq C \qquad (3)$$

taking the smallest k for which this equation holds gives x_k as a level-C lower bound for X_q. We can replace this equation by the

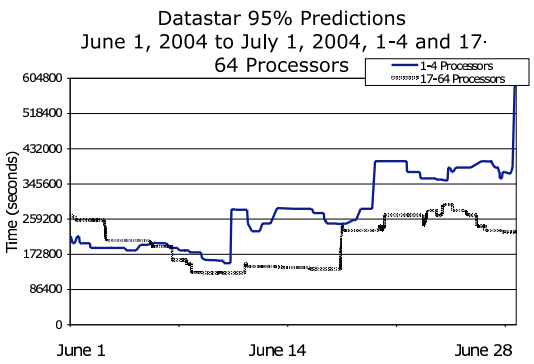

Figure 2. Predicted queue delay upper bounds on SDSC Datastar for 1-4 processors (black line) and 17-64 processors (gray line)

equivalent

$$\sum_{j=k+1}^{n} \binom{n}{j} \cdot (1-q)^{n-j} \cdot q^j \leq 1 - C \tag{4}$$

which tends to have fewer terms for high quantiles.

Even using the second form of our formula is computationally costly for large sample sizes. In this case, however, the usual normal approximation to the binomial distribution, based on the Central Limit Theorem, is quite accurate in our applications, provided that both the expected number of successes and the expected number of failures is at least 10. According to this approximation, given a Bernoulli process with probability of success p, the proportion of successes in a sample of size n will be distributed approximately normally, with mean p and standard deviation $\sqrt{\frac{p(1-p)}{n}}$. Thus the raw number of successes out of n trials has approximate distribution $N(np, \sqrt{np(1-p)})$. What this means in our case is that in order to read off a confidence bound for the q quantile of a population from a sample, we need only take the q quantile of the sample and move up a further $z^* \sqrt{nq(1-q)}$ order statistics, where z^* is the appropriate critical confidence value from the standard normal table. For example, to find a 95%-confidence upper bound for $X_{.9}$, the .9 quantile of a population X, based on a sample (x_i) of size 100, we take the .9 quantile of the data, which is $x_{(900)}$, and move $1.645 \cdot \sqrt{1000 \cdot 0.9 \cdot 0.1} \cong 15.6$ more order statistics. In order to generate a conservative estimate, we round everything up to the next integer; in this case, then, we would use $x_{(916)}$ as a reliable 95%-confidence upper bound for $X_{.9}$.

While 95% confidence may intuitively seem to give extremely conservative upper bounds for quantiles, the above example illustrates the phenomenon that the upper bound produced by the binomial method actually converges, as the sample size increases, to the quantile itself; note that the above 95%-confidence upper bound for the .9 quantile is the .916 quantile of the sample, which is a remarkably tight bound.

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

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

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

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

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

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

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

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

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

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

[13] I. Guttman. *Statistical Tolerance Regions: Classical and Bayesian*. Hafner, 1970.

[14] 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.

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

[16] 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.

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

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

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

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

[21] 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.