QBETS: Queue Bounds Estimation from Time Series *

Daniel Nurmi[1], John Brevik[2], and Rich Wolski[1]

[1]Computer Science Department
University of California, Santa Barbara
Santa Barbara, California

[2]Mathematics and Statistics Department
California State University, Long Beach
Long Beach, California

May 28, 2007

Abstract

Most space-sharing parallel computers presently operated by high-performance computing centers use batch-queuing systems to manage processor allocation. Because these machines are typically "space-shared," each job must wait in a queue until sufficient processor resources become available to service it. In production computing settings, the queuing delay (experienced by users as the time between when the job is submitted and when it begins execution) is highly variable. Users often find this variability a drag on productivity as it makes planning difficult and intellectual continuity hard to maintain.

In this work, we introduce an on-line system for predicting batch-queue delay and show that it generates correct and accurate bounds for queuing delay for batch jobs from 11 machines over a 9-year period. Our system comprises 4 novel and interacting components: a predictor based on nonparametric inference; an automated change-point detector; machine-learned, model-based clustering of jobs having similar characteristics; and an automatic downtime detector to identify systemic failures that affect job queuing delay. We compare the correctness and accuracy of our system against various previously used prediction techniques and show that our new method outperforms them for all machines we have available for study.

*The work was supported in part by NSF Grants Numbered CCF-0526005, CCF-0331654, and NGS-0305390, and by the San Diego Supercomputer Center

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 are efficiently utilized. 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.

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" 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 [28], LoadLeveler [1], EASY [20], NQS/NQE [23], Maui [22] and GridEngine [16] each offer 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 than the job's eventual execution time – for a job to begin executing. Many users often find this potential for unpredictable queuing delay particularly frustrating since, in production settings, they often *can* make reasonable 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 a method for automatically 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. The method consists of three interacting but essentially independent components: a percentile estimator, a change-point detector, and a clustering procedure. At a high level, clustering is used to identify jobs of similar characteristics. Within each cluster, job submissions are treated as a time series and the change-point detector delineates periods of stationarity. Finally, the percentile estimator computes a quantile that serves as a bound on future wait time based only on history from the most recent stationary region in each cluster. All three components can be implemented efficiently so that on-line, real-time predictions are possible. Thus, for each job submission, our method can generate a predicted bound on its delay using a stationary history of previous jobs having similar quantitative characteristics. In addition, as jobs complete their time in queue, new data becomes available. Our method automatically incorporates this information by adjusting its clustering and change-point estimates in response to the availability of new data.

The percentile estimation method we describe here is a product of our previous work in predicting the minimum time until resource failure [3, 24, 26]. In this work, we describe its application to the problem of predicting bounds on the delay experienced by individual jobs waiting for execution in batch-controlled parallel systems. To do so effectively, we have coupled this methodology with a new method for detecting change points in the submission history and a new clustering methodology that automatically groups jobs into service classes. This latter capability is necessary since many sites implement dynamically changing priority schemes that use "small" jobs to "backfill" [19] the machine as a way of ensuring high levels of resource utilization. Moreover, our quantile-based prediction method makes it possible to infer when the the machine may have crashed while the queuing system still accepts jobs (a common failure mode in these settings where jobs are submitted from one or more "head" nodes). Using this new system, we have found that it is possible to predict bounds on the delay of individual jobs that are tighter then parametric methods based on Maximum Likelihood Estimation (MLE) of Weibull, log-normal, and log-uniform distributions. To achieve these tighter bounds, however, all four components – non-parametric quantile estimation, change-point detection, clustering, and availability inference – must be integrated and employed in concert. Because the systems in inherently an adaptive time series forecasting methodology, we give it the name QBETS as an acronym for **Q**ueue **B**ounds **E**stimation from **T**ime **S**eries.

We compare QBETS with various parametric methods in terms of prediction correctness and accuracy. We also demonstrate how the combination of techniques that compose QBETS improves the predictive power for production systems.

Our evaluation uses job submission traces from 11 supercomputers (including 8 currently in operation) operated by the National Science Foundation and the Department of Energy over the past 10 years comprising approximately 1.4 million job submissions. By examining job arrival time, requested execution time, and requested node count, we simulate each queue in each trace and compute a prediction for each job. Our results indicate that QBETS (which is more effective than competitive parametric methods) achieves significantly tighter bounds on job wait

time in most cases. Thus the system automatically "reverse engineers" the *effective* priority scheme that is in place at each site and determines what job sizes are receiving the fastest turn-around time.

Thus, this paper makes two significant new contributions with regard to predicting individual job queue delays.

- We present QBETS as an example of an accurate, non-parametric, and fully automatic method for predicting bounds (with specific levels of certainty) on the amount of queue delay each individual job will experience.
- We verify the efficacy of QBETS and detail its ability to automatically take into account job resource characteristics to improve prediction bounds using currently operating large-scale batch systems, and from archival logs for systems that are no longer in operation.
- We describe an implementation of QBETS that provides an on-line batch queue job delay prediction service to high performance computing users and how we have made available a number of programmatic interfaces to the system such that others may trivially integrate QBETS into their own projects.

We believe that these results constitute a new and important capability for users of batch-controlled resources. Using an on-line, web-based, real-time version of QBETS [27] that allows users to generate predictions on demand, these users are better able to decide on which machines to use, which queues on those machines to use, the maximum amount of run time to request, and the number of processors to request so as to minimize job turnaround time or maximize the utilization of their respective time allocations. In a related work, we show how QBETS has already been used to augment a real application workflow scheduler to achieve a 2x improvement in overall workflow completion time [25]. Our techniques are also useful as a scheduling policy diagnostic for site administrators. For example, our results indicate that the amount of requested execution time is a far more significant factor in determining queue delay than is requested processor count (presumably due to back-filling [19]). One site administrator at a large scale computer center expressed surprise at this result, since she believed she had set the scheduling policy at this site to favor jobs with large processor counts in an effort to encourage users to use the resource for "big" jobs. Because short jobs can be more readily scheduled when back-filling is used, users are circumventing the site policy and submitting small jobs to improve turn-around time. In addition, we have successfully explored the use of these types of predictions to construct a "virtual resource reservation" out of regular batch queue controlled resource (*Cf.* Section 5). These examples illustrate how QBETS is already having an impact on large-scale batch-controlled settings by improving application turnaround time, streamlining large scale scheduling policies, and providing a new service to the community which has been adopted by several projects.

This ability to make predictions for individual jobs distinguishes our work from other previous efforts. An extensive body of research [6, 8, 9, 11, 12, 13, 14, 32] investigates the statistical properties of offered job workload for various HPC systems. In most of these efforts, the goal is to formulate a *model* of workload and/or scheduling policy and then to derive the resulting statistical properties associated with queuing delay through simulation. Our approach focuses strictly on the problem of *forecasting* future delay bounds; we do not claim to offer an explanatory, or even a descriptive, model of user, job, or system behavior. However, perhaps because of our narrower focus, our work is able to achieve predictions that are, in a very specific and quantifiable sense, more accurate and more meaningful than those reported in the previous literature.

In Section 2, we discuss related approaches further, followed by a detailed description of QBETS in Section 3. As mentioned previously, Section 4 discusses our predictor performance experiment, evaluation procedure and the specific results we have achieved. We briefly cover some of the ways in which the QBETS system is already impacting other existing research projects in Section 5. Finally, in Section 6 we recap and conclude.

2 Related Work

Previous work in this field can be categorized into two groups. The first group of work belongs under the general heading of the scheduling of jobs on parallel supercomputers. In works by Feitelson and Rudolph [12, 13], the authors outline various scheduling techniques employed by different supercomputer architectures and point out strengths and deficien-

cies of each. The prevalence of distributed memory clusters as supercomputer architectures has led to most large scale sites using a form of "variable partitioning" as described in [12]. In this scheme, machines are space-shared and jobs are scheduled based on how many processors the user requests and how much time they specify as part of the job submission. As the authors point out, this scheme is effective for cluster-type architectures but leads to fragmentation as well as potentially long wait times for jobs in the queue.

The second field of previous work relevant to our work involves using various models of large-scale parallel-job scenarios to predict the amount of time jobs spend waiting in scheduler queues. These works attempt to show that batch-queue job wait times can be inferred under the conditions that one knows the length of time jobs actually execute and that the algorithm employed by the scheduler is known. Under the assumption that both of these conditions are met, Smith, Taylor and Foster introduce in [32] a prediction scheme for wait times. In this work, the authors use a template-based approach to categorize and then predict job execution times. From these execution-time predictions, they then derive mean queue delay predictions by simulating the future behavior of the batch scheduler in faster-than-real time. In practice, however even when their model fits the execution-time data well, the mean error ranges from 33% to 73%.

Downey [8, 9] uses a similar set of assumptions for estimating queue wait times. In this work, he explores using 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. As a base case, Downey performs a simulation which has access to the exact execution times of jobs in the queue, plus knowledge of the scheduling algorithm, to provide deterministic wait time predictions for the job at the head of the queue. As a metric of success, Downey uses the correlation between the wait times of the head jobs during the base case simulation and the wait times experienced by head jobs if his execution time model is used.

Both of these approaches make the underlying assumption that the scheduler is employing a fairly straightforward scheduling algorithm (one which does not allow for special users or job queues with higher or lower priorities), and also that the resource pool is static for the duration of their experiments (no downtimes, administrator interference, or resource pool dynamism).

Our work differs from these approaches in two significant ways. First, instead of inferring from a job execution model the amount of time jobs will wait, we make job wait time inference from the actual job wait time data itself. The motivation for why this is desirable stems from research efforts [7, 17], which suggest that modeling job execution time may be difficult for large-scale production computing centers. Further, making inference straight from the job wait time data, we avoid having to make underlying assumptions about scheduler algorithms or machine stability. We feel that in a real world scenario, where site scheduling algorithms are rarely published and are not typically simple enough to model with a straightforward procedure, it is unlikely that valid queue wait-time predictions can be made with these assumptions.

Second, our approach differs in the statistic we use as a prediction. Most often, researchers look for an estimator of the expected (mean) wait time for a particular job. Our approach instead uses 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. For example, the information that the expected wait time for a particular job is 3 hours tells the user less about what delay his or her job will experience than the information that there is a 75% chance that the job will execute within 15 minutes.

3 Batch Queue Prediction

In this section, we describe our approach to the four related problems that we must solve to implement an effective predictor: quantile estimation[1], change-point detection, job clustering, and machine availability inference. The general approach we advocate is first to determine if the machine of interest is in a state where jobs are being serviced, next to cluster the observed job submission history according to jobs having similar quantitative characteristics

[1]We use the term "quantile" instead of the term "percentile" throughout the remainder of this paper.

(e.g. requested node count, requested maximum execution time, or requested node-hours), then to identify the most recent region of stationarity in each cluster (treated as a time series), and finally to estimate a specific quantile from that region to use as a statistical bound on the time a specific job will wait in queue. While logically the steps occur in this order, we describe them in reverse order, providing only a summarization of our quantile estimation and stationarity approaches, primarily due to space constraints but also because we have analyzed these extensively in other publications [4, 25].

3.1 Quantile Prediction

Our goal is to determine an upper bound on a specific quantile at a fixed level of confidence, for a given population whose distribution is unknown. If the quantile were known with certainty, and the population were the one from which a given job's queue delay were to be drawn, this quantile would serve as a statistical bound on the job's waiting time. For example, the 0.95 quantile for the population will be greater than or equal to the delay experienced by all but 5% of the jobs. Colloquially, it can be said that the job has a "95% chance" of experiencing a delay that is less than the 0.95 quantile. We assume that the quantile of interest (0.95, 0.99, 0.50, etc.) is supplied to the method as a parameter by the site administrator depending on how conservative she believes the estimates need to be for a given user community.

However, since the quantiles cannot be known exactly and must be estimated, we use an upper confidence bound *on the quantile* that, in turn, serves as a conservative bound on the amount of delay that will be experienced by a job. To be precise, to say that a method produces an upper 95% confidence bound on a given quantile implies that the bound produced by this method will, over the long run, overestimate the true quantile 95% of the time. The degree of conservatism we assume is also supplied to the method as a confidence level. In practice, we find that while administrators do have opinions about what quantile to estimate, the confidence level for the upper bound is less meaningful to them. As a result, we typically recommend estimating what ever quantile is desired by the upper 95% confidence bound on that quantile.

In this work, we examine the performance of four quantile prediction techniques. The first three are somewhat traditional techniques, each based on fitting a statistical distribution to historical data and using the distribution quantile of interest as the predictor for the next observation. We rely on MLE model fitting of three distributions; log-normal, log-uniform, and Weibull. We note that for the log-uniform and Weibull method, there is no straightforward way to place confidence bounds on population quantiles and thus we use the model quantile as the predictor. For the log-normal and binomial method predictors, we use the upper 95% confidence bound, but note that even when we use tighter confidence intervals, the resulting predictions are not significantly impacted. The fourth approach is a novel, non-parametric method which makes inference directly from the data, instead of assuming some pre-defined underlying distribution. Here we describe our novel method, which we term the *Binomial Method*, beginning with 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 an *upper level-C 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) holds true for observed sequences of job wait times from the real

systems, and these failures present distinct potential difficulties for our method.

Let us first (briefly) 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.

Although our method is not invalidated by dependence, 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 static 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 [2]. For the present application, however, there are a number of obfuscating factors that foil typical 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 standard time-series approaches for exploiting correlation.

3.2 History Trimming

Unlike the issue of independence and correlation, the issue of non-stationarity *does* place limitations on the applicability of quantile prediction methods. Clearly, for example, they will fail in the face of data with a "trend," say, 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 that wait-time data is typically stationary for a relatively long period and then undergoes a "change-point" into another stationary regime with different population characteristics. We thus use the Binomial Method 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 change-points in this setting.

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 change-point. Why three in a row? To borrow a well-known expression from Tukey , 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. Thus, to determine if a change-point has occurred, we compute the autocorrelation of the most recent history, look up the maximum number of "rare" events that should normally occur with this level of autocorrelation, and determine whether we have surpassed this number. If so, our

method assumes the underlying system has changed, and that the relevant history must be trimmed as much as possible to maximize the possibility that this history corresponds to a region of stationarity. Note that indiscriminate history-trimming will not allow our method to function properly, since the resulting small sample sizes will generate unnecessarily conservative confidence bounds.

The minimum useful history length depends on the quantile being estimated and the level of confidence specified for the estimate. For example, it follows from Equation 1 above that in order to produce an upper 95% confidence bound for the .95 quantile, the minimum history size that can be used is 59. (This reflects the fact that $.95^{59} < .05$, while $.95^{58} > .05$.)

3.3 Job Clustering

According to our observations and to anecdotal evidence provided by users and site administrators, there are differences among the wait times various jobs might expect to experience in the same queue, based purely on characteristics of the jobs such as the amount of time and the number of nodes requested. This is certainly easy to believe on an intuitive level; for example, if a particular queue employs backfilling [19], it is more likely that a shorter-running job requesting a smaller number of nodes will be processed during a time when the machine is being "drained." Thus, for a given job, we might hope to make a better prediction for its wait time if we took its characteristics into account rather than making one uniform prediction which ignores these characteristics.

On the other hand, the same difficulties arise in trying to produce regression models [32] as we encountered in the problem of trying to use autoregressive methods: In particular, the data are typically multimodal and do not admit the use of simple quantile prediction models. We therefore explore the idea of *clustering* the data into groups having similar attributes, so that we can use our parametric and non-parametric predictors on each cluster separately.

In fact, in [5], based on advice we received from several expert site administrators for currently operating systems, we employed a rather arbitrary partitioning of jobs in each queue by processor count, running separate predictors within each partition, which resulted in substantially better predictions. However, it would clearly be desirable to find a partition which is in some (statistical) sense "optimal" rather than relying on such arbitrary methods; for our purposes, it is also desirable to find a partitioning method that can be machine-learned and is therefore applicable across different queues with different policies and user characteristics without direct administrator intervention or tuning. Moreover, as a diagnostic tool, it would be advantageous to be able to compare the machine-determined clustering with that determined by site administrators to illuminate the effects of administrator-imposed scheduling policies. In this section, we describe our approach to this problem, which falls under the rubric of *model-based clustering* [18, 30, 35].

3.4 Model-Based Clustering

The problem of partitioning a heterogeneous data set into clusters is fairly old and well studied [18, 21, 30, 35]. The simplest and most common clustering problems involve using the values of the data, relative to some notion of distance. Often, one postulates that the distribution within each cluster is Gaussian, and the clusters are formed using some well-known method, such as the so-called k-means algorithm [21] or one of various "hierarchical" or "partitional" methods [30, 35]. If the number of clusters is also unknown, a model-selection criterion such as BIC [31], which we will discuss further below, is often used to balance goodness of fit with model complexity.

In fact, it is tempting, if for no other reason than that of simplicity, to form our clusters in this way, according to how they naturally group in terms of one or more job attributes. Note, however, that this method of clustering in no way takes into account the wait times experienced by jobs, which is ultimately the variable of interest; it is by no means clear that a clustering of jobs by how their requested wait times group will result in clusters whose wait-time distributions are relatively homogeneous. For example, it is possible that a subset of the requested job execution times form a nice Gaussian cluster between 8 and 12 minutes, but that due to some combination of administrative policy, backfilling, and various "random" characteristics of the system as a whole, jobs requesting less than 10 minutes experience substantially different wait times than those requesting more than 10 minutes, so this cluster is actually meaningless in terms of predicting wait times.

In our case, then, the situation is somewhat more complicated than ordinary clustering: We wish to cluster the data according to some characteristics which are *observable at the time the job is submitted* (explanatory variables), but using the actual wait times (response variable) as the basis for clustering. That is, we wish to use observed wait times to cluster jobs, but then to determine how each cluster is characterized by quantitative attributes that are available when each job is submitted so that an arriving job can be categorized before it begins to wait. In the discussion that follows, we will use the *requested execution time* (used to implement backfilling) as the explanatory characteristic, but this is only for the sake of ease of exposition.

The idea behind our method runs as follows: We postulate that the set of requested times can be partitioned into k clusters $C_1, \ldots, C_k$, which take the form of intervals on the positive time axis, such that within each C_j the wait times are governed by an exponential distribution with unspecified parameter λ_j.

The choice of exponential distributions is something of an oversimplification – in fact a Weibull, log-normal or hyperexponential would probably be a more accurate choice – but the fact that the clusters are relatively homogeneous makes the exponential model accurate enough with relatively little computational expense; moreover, in practice, exponentials are more than discerning enough to produce an adequate number of clusters. As a check, we generated an artificial trace using different log-normally distributed wait times corresponding to the intervals of requested times $[1, 100]$, $[101, 200]$, $[201, 300]$, $[301, 400]$, and $[401, 500]$ and fed this data to our clustering method. It recovered the following clusters for the data: $[1, 39]$, $[40, 40]$, $[41, 100]$, $[101, 197]$, $[198, 300]$, $[301, 398]$, $[399, 492]$, $[493, 493]$, $[494, 500]$. Since our method always clusters the ends together to ensure that these clusters contain at least 59 elements, the exponential clustering method recovers the original clusters almost exactly.

We assume that the appropriate clustering is into connected intervals along the time axis; this provides an intuitive model for the eventual users of our predictions. Given a desired value for the number k of clusters, then, we use a modified form of *hierarchical clustering*. According to this method, we start with each unique value for the requested time in its own cluster. We then merge the two adjacent (in the sense of adjacency on the time axis) clusters that give the largest value of the *log-likelihood function* $\log L$, calculated jointly across the clusters, according to the maximum-likelihood estimators for the exponential parameters λ_j, which are given by $\frac{\#(C_j)}{\sum_{x \in C_j} x}$. This process continues until the number of clusters is equal to k. Note that this is a well-accepted method for clustering [21, 30, 35]; however, it does not guarantee that the resulting clustering will maximize the log-likelihood over all possible choices of k clusters, even if we assume that the clusters are all intervals. This latter problem is prohibitively expensive computationally for an on-line, real-time application, even for moderately large data sets, and we are therefore forced to use some restricted method.

Each arriving job can then be categorized by identifying the cluster whose minimum and maximum requested time straddle the job's requested time.

Continuing, the question of which value of k to use is a problem in *model selection*, which recognizes the balance between modeling data accurately and model simplicity. The most generally accepted model-selection criterion is the *Bayes Information Criterion* (BIC) [31], the form of which is

$$\mathrm{BIC}(\theta) = \log L(\theta) - \frac{p}{2} \log n,$$

where θ stands for the (vector of) free parameters in the model, L is the joint likelihood function across the whole data set, calculated using the MLE for θ, p is the dimensionality of θ ($2k - 1$ in our case: the $k - 1$ break points on the time axis to define our clusters, and the k values for the λ_j, all of which are scalars), and n is the total sample size. The first term in the BIC formula should be seen as a measure of goodness of fit, while the second term is a "penalty" for model complexity (*i.e.* one with a large number of parameters). It is always true that for a less restricted model (in our case, one allowing a larger number of clusters), the $\log L$ term will be larger, so the penalty function is critical to avoid over-parameterizing. Maximizing the BIC expression over a set of proposed models has good theoretical properties and generally produces good results in practice. Thus, our clustering strategy is to specify a range of acceptable k-values; perform the hierarchical clustering described above for each of these values of k; and then calculate the BIC expression for each resulting clustering and choose the one for which BIC is greatest.

3.5 Availability Inference

Curiously, it is common for a batch queuing system to continue to accept jobs even when some form of failure has disabled those jobs from being eligible for execution on a set of computation nodes. We know of no automatic detector for this condition that is part of the production batch-scheduling systems used by the machines in our study. Moreover, based on our discussion of this issue with various site administrators, one common solution to this problem seems to be to rely on the users to call when they observe that jobs are no longer being released for execution (even though they can still be queued) and enquire as to whether there is a "problem.". If a ubiquitous service for notification of machine unavailability becomes common, QBETS can trivially be augmented to use such a system. In the meantime, we have found an elegent method to infer machine failures directly from the job waittime data.

To avoid incorporating jobs with artificially lengthened queue delays (due to machine downtime) in the history used for forecasting, QBETS attempts to infer when the computational part of the machine may be down so that these delays can be filtered. Notice that the combination of Binomial-based quantile estimation and history trimming (sans clustering) provides a relatively general non-parametric method for estimating bounds in time series. QBETS uses this generality in two ways.

First, it counts the number of jobs that have arrived between the points in time when the scheduler releases jobs for execution. As each count is generated, it is incorporated into a time series from which the upper 0.95 quantile (with 95% confidence) is estimated using a Binomial estimator with history trimming. When a count exceeds this upper bound, the QBETS predictor declares the machine to be potentially down until the scheduler releases another job for execution. This functionality is intended to mimic user behavior in which a queue that has been observed to grow "too long" indicates that the computational nodes may be unavailable.

QBETS also maintains a second upper 0.95 quantile predictor to forecast the bounds on the delay between job releases by the scheduler, again using a trimming Binomial estimator. If the time between when jobs are released exceeds the prediction of the bounds, the machine is also marked down until the next job is released. This detector is intended to reflect a user's determination that it as been "too long" since a job was released for execution.

When QBETS temporarily marks a machine as "down", jobs submitted during the down periods are not forecast. Instead, the user is given a signal that can be interpreted to mean "it is possible that the machine is down at this moment so no prediction is available." Since there is no ground truth as to when the machines in this study were actually down (no failure detector were or are available) it is impossible to know the extent to which this method generates false positive predictions. In general, however, the number of jobs for which "no prediction" would have been returned is a small fraction (usually less than 1%) of the total job submission count.

4 Results

In this section, we describe our method for evaluating the performance of our chosen batch-queue waittime prediction system, and we then detail a set of simulation experiments that take as input traces of job submission logs gathered at various supercomputing centers. We describe the details of the simulations and then report the prediction performance that users *would have* seen had the tested system been available at the time each job in each trace was submitted.

We investigate the problem in terms of estimating an upper bound on the 0.95 quantile of queuing delay; however, our approach 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 also effectively produces confidence bounds for the median (*i.e.*, the point of "50-50" probability). 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, for typical batch-queue data, which is right-skewed with a substantial tail, 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.

Note also that our assertion of retroactive prediction correctness and accuracy assumes that users

would not have changed the characteristics of the jobs they submitted in response to the availability of the quantile predictions we generate. Moreover, the on-line prototype we have developed, while operational, is in use by only a few users (in fact, we ourselves used QBETS to select which site to execute many of the simulations that generated the results reported here), making it difficult to analyze whether, and how, predictions affect workload characteristics. However, unless such feedback induces chaotic behavior, our approach is likely to continue to make correct and accurate predictions under the new conditions. We do plan to monitor the workloads experienced by various sites after the system is deployed for general use at various large-scale sites and report on the effects as part of our future work.

4.1 Data Sets

We obtained 11 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.

The job logs come from three machines operated by the San Diego Supercomputer Center during three different periods: the Intel Itanium 2 based TeraGrid cluster (**sdscteragrid**), The SDSC "Blue Horizon" (**sdscblue**) and the IBM Power-4 system (**datastar**). We also use traces from the Cornell Theory Center (**ctc**), Lawrence Livermore National Laboratory's SP-2 (**llnl**), the Cray-Dell cluster operated by the Texas Advanced Computing Center (**lonestar**), the National Center for Supercomputing Applications TeraGrid cluster (**ncsateragrid**), the California NanoSystems Institute Dell cluster (**cnsidell**), the Tokyo Tech Tsubame Supercomputer (**tsubame**), the Renaissance Computing Center (Renci) research cluster (**dante**) and the Argonne National Labs/University of Chicago TeraGrid (**ucteragrid**). The **ctc** and **sdscblue** logs we obtained from Feitelson's workload web site [10], the **llnl** data appears courtesy of Brent Gorda at LLNL, and we gathered the rest of the traces using our own infrastructure for real-time predictions. Collectively, the data comprises over one million job submissions spanning approximately a 9-year period.

4.2 Simulation

Our simulator takes as input a file containing historical batch-queue job wait times from a variety of machine/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 each of which contains one job entry per line comprising the UNIX time stamp when the job was submitted, the duration of time the job stayed in the queue before executing, the amount of requested execution time, and the node count.

The steady-state operation of the simulation reads in a line from the data file, makes a prediction (using one of the four prediction methodologies covered in Section 3) and stores the job in a "pending queue". The simulation then reads the next job arrival from the input file and, before making a prediction, potentially performs a number of tasks.

First, the simulator checks whether any jobs that had been previously queued have exited the queue since the last job arrived, in which case each such job is simply added to a growing list of historical job wait times stored in memory. Although the waiting time for the new 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 historical record changes, the predictor is given the new record so that it can update its internal state, if necessary.

After the queue has been updated, the current prediction value is used to make a prediction for the new job entering the queue, the simulation determines whether 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.

In our first set of experiments, we use only the above simulator features to make predictions for each of the jobs in our traces, varying the predictor used (binomial method, log-normal, log-uniform, and Weibull). For our second set of experiments, we add history trimming, automatic job clustering, and availability inference, as described in Section 3, in the following ways.

When a job arrives, the predictor makes a predic-

Machine/Queue	Correctness				Accuracy			
	BM	LogN	LogU	Weib	BM	LogN	LogU	Weib
cnsidell/ALL	0.92	0.97	0.97	0.81	1.00	**0.21**	**0.48**	2.14
dante/dque	0.82	0.75	0.96	0.40	1.00	1.28	**0.48**	8.82
datastar/TGnormal	0.91	0.83	0.98	0.84	1.00	4.16	**0.25**	3.51
datastar/express	0.93	0.88	1.00	0.84	1.00	3.16	**0.11**	3.90
datastar/high	0.90	0.92	0.97	0.85	1.00	0.74	**0.27**	1.48
datastar/normal	0.91	0.91	0.99	0.88	1.00	0.90	**0.17**	1.37
ucteragrid/dque	0.89	0.88	1.00	0.94	1.00	11.28	**0.00**	12.30
lonestar/development	0.92	0.92	1.00	0.92	1.00	3.30	**0.00**	4.40
lonestar/high	0.96	0.98	1.00	0.94	**1.00**	**0.61**	**0.22**	1.54
lonestar/normal	0.92	0.84	1.00	0.84	1.00	4.00	**0.04**	4.74
lonestar/serial	0.97	0.95	1.00	0.94	**1.00**	**2.77**	**0.03**	4.54
ncsateragrid/debug	0.93	0.88	0.99	0.91	1.00	2.02	**0.14**	0.59
ncsateragrid/dque	0.93	0.89	1.00	0.91	1.00	1.06	**0.06**	0.51
ncsateragrid/gpfs-wan	0.99	1.00	1.00	0.93	**1.00**	**0.16**	**0.55**	0.66
sdscteragrid/dque	0.93	0.86	0.98	0.90	1.00	2.44	**0.23**	0.26
tsubame/B	0.93	0.94	1.00	0.94	1.00	11.38	**0.00**	4.22
tsubame/default	0.93	0.84	1.00	0.84	1.00	13.16	**0.01**	6.22
tsubame/gaussian	0.96	0.94	1.00	0.95	**1.00**	137.71	**0.08**	**23.15**
tsubame/high	1.00	0.97	1.00	0.97	**1.00**	**210.66**	**0.14**	**37.95**
ctc/ALL	0.94	0.97	1.00	0.92	1.00	**0.48**	**0.04**	0.49
llnl/ALL	0.96	0.99	1.00	0.94	**1.00**	**0.29**	**0.08**	0.63
sdscblue/high	0.90	0.90	1.00	0.79	1.00	0.53	**0.15**	1.51
sdscblue/low	0.90	0.99	1.00	0.89	1.00	**0.36**	**0.11**	1.09
sdscblue/normal	0.89	0.94	1.00	0.85	1.00	0.44	**0.09**	1.13
sdscblue/express	0.92	0.90	0.99	0.84	1.00	1.12	**0.17**	2.20

Table 1: Correctness and accuracy results of four predictors without QBETS . Under **Correctness**, values >= 0.95 indicate a correct result. Under **Accuracy**, highest RMS error ratio indicates most accurate method.

tion using its current historical window as before and in addition updates the availability inference engine with the current state of the queue, which potentially changes the state of the machine to 'unavailable'. When a job in the pending queue moves into the historical window, it is passed to the predictor, which may then trim the history as previously described. Every time a pre-determined number (1000 in our study) of simulated jobs are processed, automatic clustering is performed on the entire job history.

The code implementing the simulator is modularized so that any individual component of the system (predictor, history trimming system, clustering algorithm, availability inference algorithm) can be toggled on/off or replaced at runtime. In addition, the nature of the prediction employed methodologies allow the simulator to provide an "on-line" service; meaning it can be executed in a mode where it waits in an idle state until a new job datum arrives, at which point it will update its history and refresh its predictor.

4.3 Correct and Accurate Predictions

We define a *correct* prediction to be one that is greater than or equal to a job's eventual queuing delay, and a *correct predictor* to be one for which the total fraction of correct predictions is greater than or equal to the success probability specified by the target quantile. For example, a correct predictor of the 0.95 quantile generates correct predictions for at least 95% of the jobs that are submitted.

Notice that it is trivial to specify a correct predictor under this definition. For example, to achieve a correct prediction percentage of 95%, a predictor

Machine/Queue	Correctness				Accuracy			
	BM	LogN	LogU	Weib	BM	LogN	LogU	Weib
cnsidell/ALL	0.96	0.93	0.93	0.97	**1.00**	0.05	1.55	**0.51**
dante/dque	0.80	0.69	0.87	0.72	1.00	0.05	0.71	0.40
datastar/TGnormal	0.97	0.90	0.97	0.96	**1.00**	0.46	**0.85**	**0.70**
datastar/express	0.97	0.87	0.99	0.93	**1.00**	0.78	**0.59**	1.28
datastar/high	0.96	0.95	0.98	0.95	**1.00**	**0.31**	**0.78**	**0.75**
datastar/normal	0.95	0.92	0.97	0.93	**1.00**	0.23	**0.65**	1.00
ucteragrid/dque	0.96	0.94	1.00	0.96	**1.00**	0.25	**0.19**	**0.78**
lonestar/development	0.98	0.92	1.00	0.96	**1.00**	2.12	**0.07**	**2.85**
lonestar/high	0.98	0.95	1.00	0.96	**1.00**	**0.34**	**0.29**	**0.81**
lonestar/normal	0.96	0.89	0.99	0.94	**1.00**	0.07	**0.50**	0.66
lonestar/serial	0.97	0.81	1.00	0.92	**1.00**	1.17	**0.21**	0.49
ncsateragrid/debug	0.96	0.86	0.98	0.91	**1.00**	1.37	**0.55**	2.02
ncsateragrid/dque	0.93	0.91	0.97	0.93	1.00	0.17	**0.46**	1.06
ncsateragrid/gpfs-wan	0.92	0.98	1.00	0.93	1.00	**0.38**	**0.66**	0.96
sdscteragrid/dque	0.96	0.88	0.98	0.93	**1.00**	0.12	**0.89**	0.50
tsubame/B	0.98	0.91	1.00	0.97	**1.00**	2.45	**0.29**	**1.27**
tsubame/default	0.97	0.94	1.00	0.96	**1.00**	0.05	**0.18**	**0.73**
tsubame/gaussian	0.98	0.95	1.00	0.97	**1.00**	**177.20**	**0.08**	**8.37**
tsubame/high	0.99	0.97	1.00	0.98	**1.00**	**70.46**	**0.17**	**18.76**
ctc/ALL	0.96	0.93	0.99	0.93	**1.00**	0.48	**0.18**	1.66
llnl/ALL	0.97	0.95	0.99	0.95	**1.00**	**0.65**	**0.57**	**1.58**
sdscblue/high	0.96	0.96	0.97	0.94	**1.00**	**0.24**	**0.87**	0.97
sdscblue/low	0.96	0.96	0.99	0.95	**1.00**	**0.26**	**0.38**	**1.08**
sdscblue/normal	0.97	0.95	0.97	0.95	**1.00**	**0.24**	**0.55**	**1.03**
sdscblue/express	0.97	0.91	0.98	0.94	**1.00**	0.17	**0.50**	0.55

Table 2: Correctness and accuracy results of four predictors using QBETS . Under **Correctness**, values >= 0.95 indicate a correct result. Under **Accuracy**, highest RMS error ratio indicates most accurate method.

could return an extremely large prediction (*e.g.,* a predicted delay of several years) for 19 of every 20 jobs, and a prediction of 0 for the 20^{th}. To distinguish among correct predictors, we compare their *accuracy* in terms of the error they generate, where error is some measure of the difference between predicted value and the value it predicts.

In this work, we will use Root Mean Square (RMS) error for the over-predictions as a measure of accuracy for correct predictors. We consider only over-prediction error, as we believe that the error generated for the percentage of jobs that are incorrectly predicted is relatively unimportant to the user. For example, among predictors that are 95% correct, it is our contention that users would prefer one that achieves lower over-prediction error for the 95% of the jobs it predicts correctly over one that achieves a lower error rate on the 5% that are incorrectly predicted at the expense of greater overall error in the correct predictions.

Note that one cannot compare predictors strictly in terms of their error without taking into consideration their correctness. For example, a predictor that estimates the mean of each stationary region will generate a lower RMS than one that estimates the 0.95 quantile, but the mean predictor will not provide the user with a meaningful delay bound (*i.e.,* one having a probability value attached to it). Thus, for a given job workload, we only compare predictor accuracy among those predictors that are correct.

Note also that, while RMS error is used widely as a measure of accuracy for predictions of expected values (*e.g.* in time series), its meaning is less clear in the context of quantile prediction. In this paper, we are focusing on estimating a time value which is greater than the wait time of a specific job with

probability .95. Therefore, if the distribution of wait times is highly right-skewed, a predictor may be working quite well and still have a very high RMS error. Thus, the actual *value* of the RMS error is not particularly meaningful; however, it is still useful as a means of *comparison*: For a particular set of jobs, if one correct prediction method has a lower RMS than another, then the first method, at least by this measure, produces tighter, less conservative upper bounds than the second.

4.4 Experiments

We perform two experiments in order to show the effectiveness of our prediction methods. The first experiment compares the correctness and accuracy of four different predictors for all data sets without the use of history trimming, job clustering or availability inference features. The results of this experiment are shown in Table 1. From the table, we first note that while each of the predictors is correct for some subset of the traces, the only predictor that is correct for all traces is the one based on the log-uniform distribution. Thus it might appear that the log-uniform-based method is the obvious winner for batch-queue prediction; however, upon closer inspection it becomes clear that the only reason this method is getting 95 percent or more of the predictions correct for any given trace is due to its extremely conservative individual predictions. This fact is reflected in the extremely low RMS ratios for the log-uniform method shown in Table 1 under **Accuracy**, which clearly indicates that the distance between the log-uniform predictions and the actual values is much greater than, say, the distance between the binomial method predictions and actual values for the same set of jobs. Note that in the table, bold values indicate that the shown method also was correct for that machine/queue/predictor tuple. The over-conservativeness of the log-uniform predictions is also borne out by the fact that, in general, its fraction of correct predictions is well above the target value of .95.

From the first set of experiments, we learned that there is no method that is both more correct and more accurate than the others. Our second experiment uses a combination of all of the features we have developed to improve both the correctness and the accuracy of each of the techniques. In Table 2, we show the results of the QBETS system on the same traces, varying only the predictor used during the simulation. Again, values in bold indicate machine/queue/predictor tuples which were correct. From these results, we can begin to see that the binomial method clearly stands apart from the rest in terms of both correctness and accuracy. Out of 25 traces, the binomial method was correct 22 times, which is more often than all others except for the log-uniform. Further, note that out of the 21 traces for which both the binomial and log-uniform methods were correct, the binomial was more accurate for every one of them. Additionally, overall, the binomial method was both correct and more accurate than any of the other predictors in 15 out of 25 traces; this number far exceeds the performance of any other predictor (log-normal 2/25, log-uniform 2/25, Weibull 5/25).

4.5 Correctness Analysis

Table 2 shows that when we use QBETS with the binomial-method predictor, we are able to predict bounds correctly for 95% or more individual job wait times for almost all of our traces. In this section, we explore the reasons for the effectiveness of QBETS and suggest that, for these reasons, the non-parametric approach should perform well when applied to other traces in the future.

In previous work [4], we showed that using history trimming is essential to ensure that a predictor not suffer from an inability to adjust to drastic infrequent increases in overall job queue wait times. In Figure 1, we can see the effect such drastic regime shifts have on a predictor without history trimming, and observe how trimming positively effects correctness on an example trace, the CNSI Dell cluster default queue (cnsidell/ALL). On the y-axis we show delay measured in seconds. Along the x-axis are Unix time stamps. The relatively straight line of values near the bottom of the graph depicts .95 quantile predictions made by the binomial method, but without QBETS enhancements, during a short time period. We can see that although a large number of observations lie above these predictions in the right half of the graph, there are enough relatively low values in the history that the inferred .95 quantile rises only very slowly. The other set of predictions, represented on the graph by a number of near-horizontal short segments, were made by the binomial method with QBETS over the same time period, is able to react to the shift toward longer wait times and is therefore able to produce more correct predictions. In general, this adaptivity

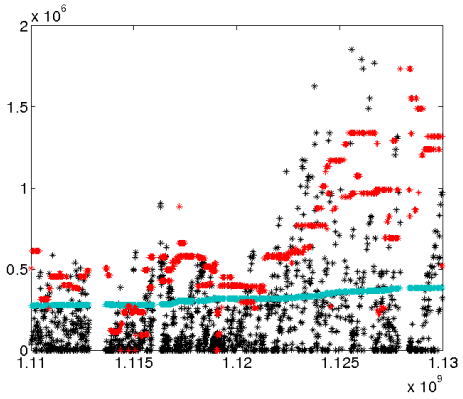

Figure 1: Job queue delay times and predictions made with and without QBETS on the CNSI Dell cluster. Dark features (black) indicate actual job wait times, the medium shaded (cyan on color displays) linear features depict predictions made without QBETS , and the light colored features (red) depict predictions made with QBETS .

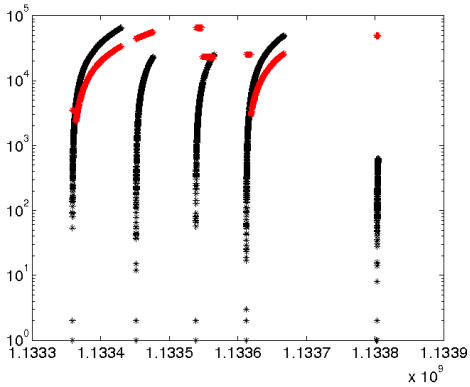

Figure 2: Actual queue delay times and QBETS binomial method predictions illustrating how frequent, drastic linear delay increases on the Dante cluster cause the method to fail. Dark features (black) show actual observed job wait times, while the light features (red on color displays) depict QBETS predictions.

greatly improves a predictor's ability to achieve its desired correctness, because such shifts are common in almost all of our traces. We note that while history trimming is an effective enhancement for all of the predictors, it works especially well with the binomial predictor; we posit that this is due to the fact that the binomial predictor is set up to make accurate inferences about quantiles, so that it is able to find changepoints in those quantiles reliably. In essence, the accuracy of the method (*Cf.* Section 4.6) feeds its correctness.

Although QBETS allows the predictor to react to drastic wait-time shifts, there are still traces for which it fails to meet the target percentage of correct predictions. In the cases where QBETS fails, we observe that in general, the reason is due to frequent drastic upward trends in wait times, which appear as 'spikes' in the trace graphs. Figure 2 shows such spikes in the middle of the Dante default queue trace. As we can see from this graph, if a large number of jobs is queued in a relatively short amount of time, and all of them experience wait times that are greater than the current quantile prediction, our method will fail to correctly make predictions for most of them, due to the fact that a wait time is not added to the predictor's available history until it comes out of the queue. Although the availability inference method attempts to discover these degenerate data cases, it cannot discover them all. In the traces for which QBETS with binomial predictor is unable to succeed, such as the dante default queue trace shown here, there are many spikes that the availability inference method does not eliminate; their negative impact on the overall correctness measure outweighs the number of jobs the method does correctly capture.

While one might be tempted to use an extremely conservative prediction method in order to combat this eventuality, this strategy may require such extreme measures as to make such a method unreasonable for non-degenerate cases. We note that even the log-uniform method, which is the most conservative method we evaluate, fails to be correct in the face of the dante default queue trace.

4.6 Accuracy Analysis

In terms of accuracy, the results presented above support two assertions. First, QBETS is the most accurate of the methods we have tested. Second, the nonparametric binomial quantile estimator is more effective than the corresponding parametric approaches. That is, when the change-point detection, clustering, and machine downtime detection features of QBETS are omitted, and we are simply applying the binomial prediction method to all jobs using the entire history, the binomial method still provides more accurate over-predictions than the other methods.

This greater accuracy, we believe, is because the binomial technique estimates directly only a specific quantile and not the entire distribution. In contrast,

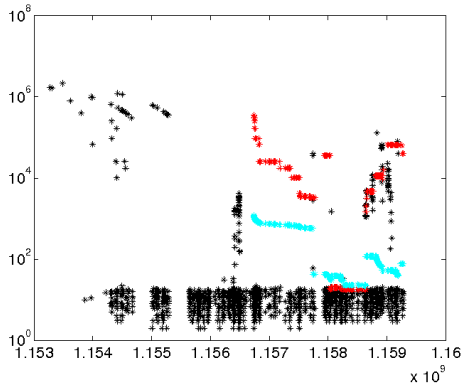

Figure 3: Trace from the Tsubame machine, Gaussian submission queue indicating large difference between log-normal and binomial method QBETS predictions after the training period. Dark features (black) show actual observed job wait times, medium shaded features (cyan on color displays) depict less conservative predictions using the log-normal, and light features (red) show predictions when the binomial method was used.

parametric approaches using MLE attempt to "fit" the data to all quantiles and in so doing may not estimate the specific quantile of interest as accurately. In particular a log-normal or Weibull model such as we have chosen to evaluate in this experiment (and typically used for such highly right-skewed data as in our traces) suffers from the fact that quantiles out in the tail of the distribution are very sensitive to the estimated population parameters. For the same reason, using a estimation technique such as MLE, the estimated parameters are sensitive to a few very high values in the data set. Thus an estimated quantile for such a distribution is highly dependent on the model's ability, typically based on a small number of high values in the data, to fill in its right tail. In practice, the end result of this phenomenon is usually that the quantile estimates produced by these parametric models are much more conservative than the ones that can be made using the binomial method, which does not need to take into account the relationship between high and (irrelevant for our purposes) low values in the way that curve fitting does.

One fundamental reason for the superior accuracy of predictions generated using QBETS stems from the automatic job-clustering feature, which allows the predictor to only consider "like jobs" when making its prediction instead of all jobs, which may be only loosely related to the job of interest in terms of experienced wait time. During the experiment, we observed that QBETS automatically grouped jobs into three to five clusters, never choosing only one group for all jobs. Additionally, we observe that not only is QBETS more correct in general, but that QBETS with the binomial method predictor outperforms the other predictors in most of the traces. Again, the reason this is true is due to the fact that in general the binomial method is making more accurate predictions, as we see from Table 2 and Table 1; this amounts to heightened sensitivity to change-points in the data, thus allowing the history-trimming feature to activate more often than it does for other predictors.

This being said, there are a few cases where the parametric models were in fact **more** accurate than the binomial method. In these cases, most notably the tsubame/guassian and tsubame/high traces, we observe that the primary reason why the log-normal is achieving so much better RMS errors stems from the fact that in those traces, the training period data included a disproportionate number of very large wait times relative to the experimental set. The training set can be seen in Figure 3 as the period of observations before any predictions are being made; notice that the binomial method starts out making very conservative predictions based on the large number of high values in the training set, while there are enough low values to bring down the MLE log-normal parameters, making these predictions less conservative. In this case, data for the training period was bimodal, with about 10% of the wait times in an extremely high mode, orders of magnitude higher than the bulk of the wait times in the lower mode. This higher mode, which would have caused the log-normal predictions to be incorrect, disappeared at the end of the training period, leaving the binomial method with an unrepresentative data set to begin with and also rendering the log-normal predictions both correct and accurate. We note two things, however: First, the experimental set was only slightly larger than the training set, so that there was not time to balance the anomalies in the training data, and so may not have been reflective of long-term performance; second, by the middle of the experimental set, the binomial method predictor was able both to make more accurate predictions than the log-normal predictor for the relatively short wait times and also to maintain correctness when the wait times suddenly became longer again at the end of the trace.

5 QBETS Impact

Currently, QBETS is providing predictions to a growing base of HPC researchers and users around the world. Our batch queue monitoring sensors are gathering real-time batch queue delay data from 16 super-computers, 24 hours a day. From this database of job delay information, the QBETS prediction software is able to constantly generate up-to-date quantile predictions through a number of interfaces. Over the past several months, our records indicate that the QBETS system has been accessed over 50000 times from approximately 600 unique, non-searchbot Internet hosts. This level of activity indicates that users interested in integrating real-time QBETS predictions into their projects are using a number of interfaces, including a C API (in the form of a UNIX library), UNIX command line tools (for curious users and administrative scripting), a dedicated QBETS Web Service (for integration into existing Web Service based projects), and our own custom QBETS web site [27]. Using these interfaces, researchers have been able to use QBETS to accomplish a number of tasks, including the provision of HPC site selection hints for users (TeraGrid User Portal [33]), in-advance workflow scheduling for disaster recovery applications (LEAD Project [29]), redundant batch queue resource provisioning for fault-tolerant systems (LEAD/VGrADS [34]). Finally, we ourselves are building on the availability of QBETS by implementing a new system for making virtual advance reservations. Recently, we have run experimental trials that show our system is capable of providing users the ability to request an advance reservation, and probabilistically servicing such requests using regular batch controlled resources, without modification to the underlying batch queue software or administrative policies. As the popularity of QBETS continues to grow, we expect to add more systems to the infrastructure and possibly even integrate our work into existing batch queue resource managers to make QBETS part of many default HPC software installations.

6 Conclusions

Space-shared parallel computers use queuing systems for scheduling parallel jobs to processor partitions in such a way that each job runs exclusively on the processors it is given. In previous work [4, 5] we have proposed a method for estimating bounds on the queuing delay experienced by each each job and show that this non-parametric method (termed the Binomial Method) outperforms competitive approaches.

Still, while working with traces comprising some 1.4 million jobs from 11 supercomputer sites which, we observed several features of the data that have a negative effect on the performance of all of our prediction methods. First, the data can exhibit what might be called long-term non-stationary, in the sense that there are infrequent events (possibly changes in policy or other fundamental changes to the operation of the queue) that have a substantial and lasting effect on queueing delays. Second, we observe that while some jobs request small allocations, other allocations are much larger. Such a situation leads to a predictor making very conservative estimates for the small jobs since we are concerned with upper bound quantile predictions. It seems that jobs may fall into "groups" of like jobs that one might expect to experience roughly similar queue wait-time delays. Finally, we note that in several of our traces, we notice sudden "spikes" in wait time delay, expressed by a drastic increase in job delay experienced in a very short time, likely indicating that the machine of interest is experiencing a period of unavailability.

We therefore introduce QBETS , which combines history trimming, automatic job clustering, availability inference, and various prediction methodologies to provide a batch queue job wait time prediction system which is shown to perform better than more naive approaches for almost all of the data we have access to. Additionally, we show that QBETS , with the non-parametric binomial method quantile predictor invented in our previous work, is both more correct and more accurate than any other tested technique and prediction method.

In the future, we intend to continue to improve both the correctness and accuracy of QBETS by exploring alternative clustering and prediction techniques and applying them experimentally to our ever-growing set of machine traces. Additionally, we intend to continue to provide real-time batch-queue wait-time predictions to the HPC user community through continued involvment in a wide variety of projects, and through our own batch queue prediction service oriented web site. Finally, we intend to use many of the techniques presented in this work towards defining a functionally static resource definition out of highly dynamic, heterogeneous underlying compute resources.

References

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

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

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

[4] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPoPP 2006*, March 2006.

[5] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay in space-shared computing environments. In *Proceedings of IEEE International Symposium on Workload Characterization 2006*, October 2006.

[6] S.-H. Chiang and M. K. Vernon. Dynamic vs. static quantum-based processor allocation. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science Vol. 1162*. Springer-Verlag, 1996.

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

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

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

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

[11] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science Vol. 1162*. Springer-Verlag, 1996.

[12] D. G. Feitelson and L. Rudolph. Parallel job scheduling: Issues and approaches. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science Vol. 949*. Springer-Verlag, 1995.

[13] D. G. Feitelson and L. Rudolph. Towards convergence in job schedulers for parallel supercomputers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science Vol. 1162*. Springer-Verlag, 1996.

[14] E. Frachtenberg, D. G. Feitelson, J. Fernandez, and F. Petrini. Parallel job scheduling under dynamic workloads. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science Vol. 2862*. 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] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[19] D. Lifka. The anl/ibm sp scheduling system. In *Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.)*, volume 949, pages 295–303. Springer-Verlag, 1995.

[20] 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.

[21] J. MacQueen. Some methods for classification and analysis of multivariate observations. pages 281–297, 1967.

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

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

[24] 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.

[25] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of SC06*, November 2006.

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

[27] NWS Batch Queue Ppreddiction web interface. `http://nws.cs.ucsb.edu/ewiki/nws.php?id=Batch+Queue+Prediction`.

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

[29] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. Mclaughlin, R. Wilhelmson, S. Graves, M. Ramamurhty, R. D. Clark, S. Yalda, D. A. Reed, E. Joseph, and V. Chandraeskar. CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting. *IEEE Computer*, 39:56–64, Nov. 2006.

[30] C. Posse. Hierarchical model-based clustering for large datasets. *Journal of Computational and Graphical Statistics*, 10(3):464–??, 2001.

[31] G. Schwartz. Estimating the dimension of a model. In *Ann. of Statistics*, pages 461–464, 1979.

[32] 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.

[33] TeraGrid user portal. `http://portal.teragrid.org`.

[34] The virtual grid application development software (vgrads). `http://vgrads.rice.edu/`.

[35] S. Z. Zhong. A unified framework for model-based clustering. *Journal of Machine Learning Research 4 (2003) 1001-1037.*