Parallel Processing Letters, Vol. 17, No. 1 (2007) 21–46
© World Scientific Publishing Company

Using Model-based Clustering to Improve Predictions for Queueing Delay on Parallel Machines

John Brevik, Daniel Nurmi, Rich Wolski
Dept. of Computer Science, University of California Santa Barbara
Santa Barbara, California, United States

Received November 2006
Revised January 2007
Communicated by Guest Editors

ABSTRACT

Most space-sharing parallel computers presently operated by production high-performance computing centers use batch-queuing systems to manage processor allocation. In many cases, users wishing to use these batch-queued resources may choose among different queues (charging different amounts) potentially on a number of machines to which they have access. In such a situation, the amount of time a user's job will wait in any one batch queue can be a significant portion of the overall time from job submission to job completion. It thus becomes desirable to provide a prediction for the amount of time a given job can expect to wait in the queue. Further, it is natural to expect that attributes of an incoming job, specifically the number of processors requested and the amount of time requested, might impact that job's wait time.

In this work, we explore the possibility of generating accurate predictions by automatically grouping jobs having similar attributes using model-based clustering. Moreover, we implement this clustering technique for a time series of jobs so that predictions of future wait times can be generated in real time.

Using trace-based simulation on data from 7 machines over a 9-year period from across the country, comprising over one million job records, we show that clustering either by requested time, requested number of processors, or the product of the two generally produces more accurate predictions than earlier, more naive, approaches and that automatic clustering outperforms administrator-determined clustering.

Keywords: Model-based clustering, queueing delay, batch-queuing, wait time prediction, trace-based simulation, parallel systems.

1. Introduction

Typically, high-performance multi-processor compute resources are managed using *space sharing*, a scheduling strategy whereby 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 (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 [22], LoadLeveler [1], EASY [18], NQS/NQE [21], Maui [20] and GridEngine [14] 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 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 *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. Unpredictable queuing delays also make program debugging, particularly for problems that occur only at scale, tedious and difficult.

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" machine. Before describing our method, we recall that for a given statistical distribution and real number α between 0 and 1, the α *quantile* of the distribution is the smallest value such that the fraction α of the population is less than or equal to that value; the α quantile can be expressed as the 100α *percentile*, but the language of quantiles is more convenient for us. The key observation for our purposes is that a randomly chosen element of a population has probability α of being less than or equal to the α quantile of that population's distribution.

Our method consists of three interacting but essentially independent components: a quantile estimator, a change-point detector, and a clustering procedure.

[*]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 resources become available.

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 regions of stationarity. Finally, the quantile 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. For example, if the quantile estimator is parameterized to predict the .95 quantile, the resulting prediction is a value greater than or equal to 95 percent of the population.

For the remainder of this work, we use the term "region of stationarity" to mean a period of time where the underlying distribution of wait times remains relatively constant, and the term "change-point" is defined as the point between regions of stationarity. For example, a machine might exhibit short wait times over holiday breaks, but when the break is over, the machine suddenly and drastically exhibits very long wait times as the load increases. In such an example, there would be two regions of stationarity (before and after the holiday break), with one change-point in between.

All three components (quantile estimator, change-point detector, and clustering procedure) 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 previous work [3] we have investigated various methods for quantile estimation and explored the efficacy of automatic change-point detection in such highly correlated data. Using a simple method based on binomial distributions, combined with on-line autocorrelation analysis, 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 and log-normal distributions. This methodology we term the *Binomial Method Batch Predictor* (BMBP) [3]. These previous results advanced the state-of-the-art in batch-queue prediction capability in two ways. First, BMBP outperforms the best competitive prediction techniques. The problem of predicting batch-queue delay is one that has been extensively studied (*cf.* Section 2), yet no approach is widely accepted as successful. Second, BMBP is the first technique to be able to provide "soft" statistical guarantees of bound on delay.

In this work, we focus on improving the bounds achieved by BMBP through a novel approach to automatic model-based clustering applied hierarchically to the job submission history it uses. While we have verified that BMBP is statistically correct, the "tightness" of the bounds it produces can be improved by categorizing jobs by their requested resource requirements. Typically, batch schedulers employ queuing policies that attempt to maintain resource utilization while preserving fairness. For example, many sites employ some form of *backfilling* [17] in which users specify maximum run times and processor counts for the jobs they submit. The scheduler maintains priority order for "large" jobs requiring many processors. Because job execution time is highly variable [15], however, to run a large job, the scheduler must collect processors as they become idle and hold them until enough are available to initiate execution. Using backfilling, the scheduler sets a deadline for initiation of a large job, and runs smaller jobs on the processors that are being collected (and would otherwise be idle) that will complete before the deadline. Thus processors remain more fully utilized and smaller jobs do not starve larger ones. Users in backfilling settings, typically, know they can "squeeze" in short jobs, but the exact parameters that distinguish "short" from "long" jobs (set by the site administrator) is not advertised. Moreover, as site priorities change, administrators are prone to "tuning" the scheduling policies, often without notifying the user community.

We describe a new methodology for automatically inferring a categorization of jobs based on their requested execution time and processor requirements, and then using that categorization to improve the tightness of the bounds we predict. We detail the full methodology (Binomial quantile estimator, change-point detector, and clustering algorithm) and discuss the often complicated interaction between change-point detection and re-clustering in response to newly available queue delay measurements when the system is to be used in an on-line setting. Thus, our goal is to explore the effect of automatic model-based clustering on the accuracy of our batch queue wait time bound predictor.

To verify the effectiveness of the approach, we compare BMBP with and without clustering, using job submission traces from 7 supercomputers (including three currently in operation) operated by the National Science Foundation and the Department of Energy over the past 9 years comprising approximately 1.2 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 BMBP (which is more effective than competitive parametric methods) used with clustering achieves significantly tighter bounds on job wait time in most cases. Thus, this new combination represents the most accurate available method for predicting bounds on individual job delay times, and does so with a specifiable degree of certainty.

Thus, this paper makes three significant new contributions with regard to predicting individual job queue delays in production batch system settings.

- We present BMBP (briefly) enhanced by a job clustering technique 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 these techniques using job submission logs from currently operating large-scale batch systems, and from archival logs for systems that are no longer in operation.
- We find that clustering improves the accuracy of the bounds, that requested execution time is a more significant factor for clustering jobs than is processor count, and that jobs cluster differently from the way expert site administrators (who control the specific scheduling policies) have anticipated.

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 BMBP (`http://nws.cs.ucsb.edu/batchq`) with clustering 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. Our techniques are also useful as a scheduling policy diagnostic for site administrators. For example, our results indicate that for some systems the amount of requested execution time is more significant factor in determining queue delay than is requested processor count (presumably due to back-filling [17]). 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 (and to disfavor short, small jobs) 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. This example illustrates how prototype versions of cluster enhanced BMBP are already having an impact in large-scale batch-controlled settings. We discuss the nature of this impact further in Section 4.

This ability to make predictions for individual jobs distinguishes our work from other previous efforts. An extensive body of research [4,6,7,9,10,11,12,25] 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, and/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. We discuss related approaches further in Section 2.

The next section discusses previous and related approaches to characterizing and predicting queue delay. In Section 3 we describe BMBP briefly and the clustering methodology in detail. As mentioned previously, Section 4 discusses our evaluation procedure and the specific results we have achieved, Finally in Section 5 we recap and conclude our description of this work.

2. Related Work

Previous work in this field can be categorized into three groups. The first group of work belongs under the general heading of scheduling jobs on parallel supercomputers. In a work by Feitelson and Rudolph [10,11], the authors outline various scheduling techniques employed by different supercomputer architectures and point out strengths and deficiencies 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 [10]. 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 group of relevant previous work involves using various models of large-scale parallel jobs to predict the amount of time jobs spend waiting in scheduler queues. These works examine whether such predictions are possible under the assumption of perfect knowledge about the length of time that jobs actually execute and about the algorithm employed by the scheduler. Smith, Taylor, and Foster [25] 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. However, under the above assumptions, the mean error in their predictions ranges from 33 to 73 percent, even when the job execution times are well modelled. Downey [6,7] uses a similar set of assumptions for estimating queue wait times. In this work, he 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. As a metric of success Downey uses the correlation between the wait time of the head job, if execution times are estimated using his model, and the head job wait time if the execution time is exactly known. 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 the above 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 this approach stems from research efforts [5,15], which suggest that modelling job execution time may be difficult for large-scale production computing centers. Further, by making inference directly from the job wait time data, we avoid having to make underlying assumptions about scheduler algorithms or machine stability; in the real world, where site scheduling algorithms are rarely published and are not typically simple enough to admit a straightforward model, it is unlikely that useful queue wait-time predictions based on these assumptions can be made.

Our approach is also distinguished by the statistic we use as the basis for prediction. Most often, we see researchers making predictions for the mean (expected) value for the queue wait time. Our approach instead uses confidence bounds on the time an individual job will wait. 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" how likely it is that their job's wait time will fall outside the range.

3. BMBP and Job Clustering

In this section, we describe our approach to the three related problems that we must solve to implement an effective predictor: quantile estimation, change-point detection, and clustering. The general approach we advocate is first to cluster the observed job submission history according to jobs having similar quantitative characteristics (*e.g.* requested node count, requested maximum execution time, or requested node-hours), next 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 the steps occur in this order in practice, we describe them in reverse order for ease of exposition; also, we provide only a summary of our approaches to quantile estimation and change-point detection, primarily due to space constraints but also because we have analyzed these extensively in a previous publication [3].

3.1. Inference for Quantiles Using the Binomial Method

We describe a simple method for determining 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 the a given quantile implies that the bound produced by this method will, in the long run, overestimate the true quantile 95% of the time. This confidence level corresponds to the desired degree of conservatism and is supplied to the method as a parameter. 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 whatever quantile is desired by the upper 95% confidence bound for that quantile.

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

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

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

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.,*[13]. 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 the time-series character of the data, 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, time-series analysis and modeling is primarily focused on using dependencies between measurements to improve forecasting [2]. For the present application, however, there are a number of factors that foil typical time-series methods. First of all, for a given job entering a queue, there are typically several jobs already in the queue, so 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 much more weight in their tails than exponential distributions; additionally, many queues exhibit bimodal or multimodal behavior. 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, neither is it amenable to standard time-series approaches for exploiting correlation.

3.2. Correct and Accurate Predictions

Because we are predicting a probabilistic bound on the delay for each job, it is useful to differentiate between a correct prediction and an accurate one in this context. We define a *correct* prediction to be one that is greater than or equal to a job's eventual queueing 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 construct a correct predictor under this definition. For example, to achieve a correct prediction percentage of 95%, a predictor 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}. Since many correct predictors are possible, we need a means for comparing them. We thus define the *accuracy* of such a predictor in terms of the error it generates, 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 because 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.

Note that using this method to compare *any* two predictors, without consideration of their correctness, is not particularly meaningful. 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 accuracy among correct predictors.

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 that first method is preferable in terms of producing tighter, less conservative upper bounds (*cf.* Section 4 for further discussion of RMS error).

3.3. Non-stationarity and Change-Point Analysis

Unlike the issue of independence and correlation, the issue of non-stationarity *does* place limitations on the applicability of our method. Clearly, for example, it will fail in the face of data with a "trend" such as a mean value that increases linearly with time. On the other hand, insisting that the data be stationary is too restrictive to have any practical value: 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, or, to look at it another way, 1 error in 20, 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.

Next, 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 [13], 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 $.95^{58} > .05$.)

Again, a more complete description of the Binomial Method and its concomitant procedure for change-point detection, as well as a more detailed rationale and analysis of the assumptions upon which it is based, are available in [3], as is evidence of its effectiveness in comparison to other methods. Here we endeavor only to summarize the approaches, which we will heretofore refer to together as the Binomial Method Batch Predictor (BMBP), and provide a general motivation for their effectiveness.

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

3.4. Prediction with Model-Based Clustering

According to our observations and to anecdotal evidence provided by users and site administrators, there are differences, arising purely from characteristics of the jobs such as the amount of time and the number of nodes requested, among the wait times various jobs might expect to experience in the same queue. This is certainly intuitively plausible; for example, if a particular queue employs backfilling [17], it is more likely that a shorter-running job requesting a smaller number of nodes can 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 [25] as we encountered in the problem of trying to use autoregressive methods: In particular, the data are typically multimodal and do not admit of simple parametric models. We therefore explore the idea of *clustering* the data into groups having similar attributes, so that we can use our non-parametric predictor on each cluster separately.

In fact, 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* [16,23,29].

3.5. Model-Based Clustering

The problem of partitioning a heterogeneous data set into clusters is fairly old and well studied [16,19,23,29]. The simplest and most common clustering problems involve grouping data values relative to some notion of distance. Often, one postulates that the distribution within each cluster is Gaussian and then forms clusters using some well-known method such as the so-called k-means algorithm [19] or one of various "hierarchical" or "partitional" methods [23,29]. If the number of clusters is also unknown, a model-selection criterion such as BIC [24], which we will discuss further below, is often used to balance goodness of fit with model parsimony.

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 [19,23,29]; 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) [24], 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.

3.6. Change-point Detection and Clustering

There is a potential difficulty implementing re-clustering as new job delay information becomes available. When the method finds a new clustering, it does not take into account the minimum necessary history required by BMBP (calculated, using the quantile of interest and desired confidence level, from Equation 1). As a result, clusters determined by the method may not be suitable for use by BMBP. We address this problem in the following way. When BMBP is required to make a forecast from a cluster that does not contain enough data, it augments the history it considers by adding the data from the next *higher* cluster in the adjacency list, adding histories from further higher clusters if necessary until enough history is available. This temporary merge is recomputed each time a forecast from a short cluster is needed, so that it is only done for a given cluster until that cluster has enough data. We term this process *history borrowing*. Borrowing is done from *higher* adjacent clusters in the interest of safeguarding the correctness of the predictor, possibly at the expense of generating slightly over-conservative estimates.

4. Results

In this section, we describe our method for evaluating BMBP with clustering, and we then detail a set of simulation experiments that use, as their input, traces of job submission logs gathered at various supercomputing centers. While we have implemented and deployed a prototype of BMBP at a number of nationally accessible large-scale sites, and we are in the processes of enhancing this prototype with clustering, a rigorous comparison is best served by repeatable, trace-based simulation. We describe the details of the simulations (we use a single simulator that can parse each job log) and then report the prediction performance users *would have* seen had BMBP and/or BMBP with clustering 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 estimates confidence bounds for the median (*i.e.*, the point of "50-50" probability) with equal effectiveness. We note that the quantiles at the tail of the distribution corresponding to rarely occurring but large values are more variable, hence more difficult to estimate, than those nearer the center of the distribution. Thus, in a typical batch queue setting, which is characterized by large numbers of jobs experiencing short wait times and a few jobs experiencing long wait times, the upper quantiles provide the greatest challenge for a prediction method. By focusing on an upper bound for the .95 quantile, we are testing the limits of what can be predicted for queue delay.

Note also that our results are trace-based and thus assume 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 [‡] making difficult an analysis of whether BMBP predictions affect workload characteristics. However, unless such feedback induces chaotic behavior resulting in frequent "spikes" in delay (*cf.* Subsection 4.5 below), BMBP is likely to continue to make correct and accurate predictions. We do plan to monitor the workloads experienced by various sites after BMBP is deployed for general use at various large-scale sites and report on the effects as part of our future work.

4.1. Simulation

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, the duration of time the job stayed in the queue before executing, the amount of requested execution

[‡]We count ourselves in the category of BMBP users, as we used the on-line system to schedule the simulations we executed for this paper at various sites where our system is currently operating.

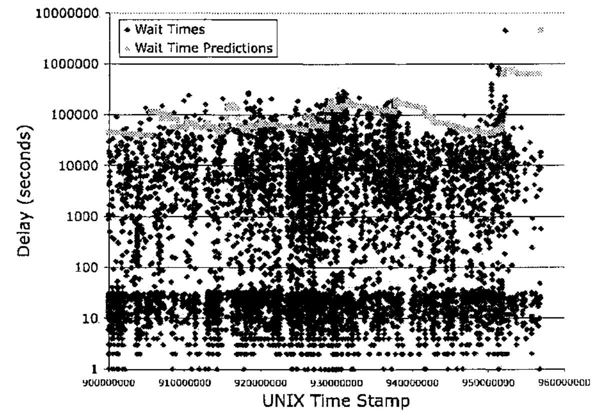

Fig. 1. Example output of BMBP simulator without the use of clustering or static grouping.

time, and the node count.

The steady-state operation of the simulation reads in a line from the data file, makes a prediction (using BMBP with and without clustering) and stores the job in a "pending queue". We then increment a virtual clock until one of two events occur:

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

When the first case occurs, the job is simply added to a growing list of historical job wait times 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 historical record changes, BMBP is given the new record so that it can update its internal state. If clustering is in use, the BMBP predictor(s) for the cluster(s) into which the jobs are placed are updated.

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.

Notice also that it may be possible to use information about waiting jobs to improve prediction quality, but our methodology does not do so; BMBP, both with and without clustering, only updates its prediction state with new job information when a job is released from the pending queue. We deliberately eschewed the use of job waiting information to guard against the possibility that job cancellations would perturb our results in a real-world setting. For example, if live predictions are available for a variety of machines, users might choose to cancel waiting jobs (or to submit multiple jobs) based on BMBP readings. Since jobs in queue are not considered, this potential user feedback response will not impact prediction behavior directly. It is possible that excessive job cancellations will affect some batch-schedulers based on whether the policies in use consider queued jobs when making scheduling decisions. We leave an *in situ* investigation of this potential secondary effect for future work.

In addition to events which occur with respect to time, one final simulation event, re-clustering, occurs after a pre-determined number (1000 in our study) of simulated job arrivals have occurred. That is, the simulator presents new delay measurements to BMBP when the simulated waiting times have expired, records forecast accuracy, and triggers re-clustering. The code implementing BMBP and BMBP with clustering are modularized so that the same implementation can be operated by the simulator or by the operational infrastructure that makes "live" job predictions in real time. The prototype is currently monitoring several active supercomputers and large-scale cluster systems. We have developed a web-based interface which can be accessed via `http://nws.cs.ucsb.edu/batchq` as well as via a set of command-line tools.

As an example, in Figure 1 we show the time series of job queue delays and 0.95 quantile predictions generated by BMBP without clustering for the SDSC SP-2 machine and the *high* queue. On the y-axis using a log scale we show delay measured in seconds. Along the x-axis are Unix time stamps. Each dark-colored graph feature represents the queue delay experienced by a particular job, and the light-colored features near the top represent the predictions.

In contrast, Figure 2 shows the same data as does Figure 1 in dark graph features, but the 0.95 quantile predictions generated by BMBP and BMBP with clustering using requested execution time as a response variable as light graph features. Note that although it may appear as though there are multiple predictions for any given points in time, each job can only belong to one cluster, and therefore the graph is composed of unique job wait time/prediction pairs.

Notice that many of the predictions shown in Figure 2 are almost an order of magnitude lower than the predictions for the corresponding point in time in Figure 1. Thus if the enhanced BMBP-determined clusters correctly categorize jobs by their response variables, BMBP applied to each cluster should yield correct predictions that are more accurate (produce a "tighter" bounds) than without clustering.

4.2. Data Sets

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.

The job logs come from three machines operated by the San Diego Supercomputer Center during three different periods: the IBM SP-2 (**sdsc**), The SDSC "Blue Horizon" (**sdscblue**) and the IBM Power-4 system (**datastar**). We also use traces

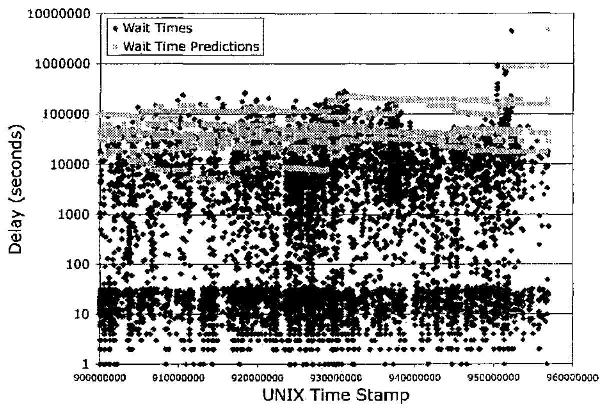

Fig. 2. Example output of BMBP/clustering simulator when using the requested time to cluster jobs and make predictions.

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**), and the Argonne National Labs/University of Chicago TeraGrid (**ucteragrid**). The **ctc**, **sdsc**, and **sdscblue** logs we obtained from Feitelson's workload web site [8], the **llnl** data appears courtesy of Brent Gorda at LLNL, and we gathered the **datastar**, **lonestar**, and **ucteragrid** traces using our own infrastructure for real-time predictions. Collectively, the data includes over one million job submissions spanning approximately an 9-year period. The site and queue names, the durations covered by each job log, and the number of predictions we make for each queue is shown in Table 1. Within each log, each job is represented uniquely by four values: submission time, queue wait time, number of nodes requested and number of seconds requested.

In addition, we contacted several site administrators, who have graciously allowed us to install and experiment with the BMBP real-time prediction system. After many consultations, the administrators furnished us with a scheme for node partitioning in the currently active systems that they believe would improve BMBP. That is, we asked each administrator to give us a static node clustering to use for his or her site. After several iterations, we converged on a single static clustering of 1 to 4 nodes, 5 to 16 nodes, 17 to 64 nodes, and greater than or equal to 65 nodes, which we currently use for the real-time prediction system.

4.3. Correctness and Accuracy Results

In the following two tables we summarize the performance of BMBP with and without clustering. Each table compares the performance of "plain" BMBP (without clustering), BMBP applied to the "static" node clustering obtained from ex-

Machine/Queue	Start Date	End Date	Pred Count
ctc/all	Jun 1996	May 1997	76206
datastar/express	Apr 2004	Apr 2005	18242
datastar/high	Apr 2004	Apr 2005	6781
datastar/normal	Apr 2004	Apr 2005	63751
datastar/TGnormal	Apr 2004	Apr 2005	6091
llnl/all	Jan 2002	Oct 2002	54953
lonestar/normal	Jan 2004	Mar 2005	27486
lonestar/serial	Jan 2004	Mar 2005	2181
sdscblue/express	Apr 2000	Dec 2002	66320
sdscblue/high	Apr 2000	Dec 2002	15781
sdscblue/low	Apr 2000	Dec 2002	16104
sdscblue/normal	Apr 2000	Dec 2002	47407
sdsc/express	Apr 1998	Apr 2000	3985
sdsc/high	Apr 1998	Apr 2000	7794
sdsc/low	Apr 1998	Apr 2000	21126
sdsc/normal	Apr 1998	Apr 2000	29765
ucteragrid/dque	Jan 2004	Oct 2005	58163

Table 1: Machine and queue names, start and end dates for each log, and the number of predictions made from each log.

pert administrators, and BMBP when clustering by requested "nodes", maximum execution time, abbreviated "rtime", and the product of the two (node-seconds) respectively, abbreviated "rnprod."

In Table 2 we show the name of each site in the first column, the queue name in the second column, and the percentage of correct predictions (rounded to two digits) in the remaining 5 columns.

Table 2 confirms the correctness of both methods, but says little about their utility. Recall that it is trivial to construct a bounds predictor that will yield exactly a 95% correctness metric, thus, to compare the effectiveness of predictors, we also measure the degree of over-prediction generated by each as a measure of accuracy.

In Table 3 we compare the RMS (root mean square) over-prediction error of BMBP to BMBP with clustering [§] The number in each cell of the Table is calculated as the ratio of RMS over-prediction by plain BMBP to that of BMBP with clustering, thereby normalizing the values such that the "plain" BMBP entry is always 1. As a result, larger numbers in the table reflect predictions that are more accurate in our sense. In each row we boldface the largest ratio as a way of denoting the most accurate methodology.

From Table 3 it can be seen that for each queue, at least one automatically determined clustering improves, or sustains, the prediction accuracy of the "plain"

[§]We realize that the value of this comparison is predicated on the assumption that BMBP is, itself, a good predictor. In [3] we verify this claim extensively, and thus we omit a re-exposition here in favor of evidence for the use of clustering to improve these predictions.

Machine/Queue	plain	split	nodes	rtime	rnprod
ctc/all	95	95	95	96	*94*
datastar/express	97	97	96	96	97
datastar/high	95	95	*94*	95	95
datastar/normal	95	95	95	95	95
datastar/TGnormal	97	97	97	97	97
llnl/all	96	96	96	96	96
lonestar/normal	96	97	96	97	97
lonestar/serial	98	97	96	98	96
sdscblue/express	*94*	95	96	95	96
sdscblue/high	97	97	97	97	97
sdscblue/low	96	95	97	96	*94*
sdscblue/normal	96	95	96	96	96
sdsc/express	98	97	98	98	97
sdsc/high	97	97	97	97	97
sdsc/low	96	96	96	96	*94*
sdsc/normal	97	96	96	96	96
ucteragrid/dque	97	97	97	97	97

Table 2: The correctness percentage for "plain" BMBP, static splitting by node count, and automatic clustering using each of three variables.

method. It is also clear that the best clustering variable varies by machine and queue, although in every case except one some form of clustering generates at least a small improvement.

4.4. Analysis

The results in Table 3 indicate a variety of scheduling policies employed by different site administrators at different times, which is perhaps, in retrospect, unsurprising. In developing BMBP with clustering, we interviewed several site administrators and many users of the machines we analyzed (except for the CTC machine whose vintage made such interviews problematic).

Administrators believed that our work would show that node count would impact job scheduling most significantly, while users were almost uniformly certain that requested execution time was the most important factor to consider. Both appear to be correct for some machine and queue combinations. For example, Lonestar in the *normal* queue and the SDSC SP-2 in the *high* queues, node count appears to be the best variable for clustering. In the Datastar *normal* and the SDSC blue *high* queues, however, requested run time (possibly due to backfilling) has the greatest impact on wait time. To explore these effects in combination, we also clustered based on the product of node count and requested time, which performed consistently well. In the Table, 7 of the 17 cases are most improved by the product, compared to 6 cases where node count is best, and 5 cases where requested execution time is best (including two ties).

Machine/Queue	plain	split	nodes	rtime	rnprod
ctc/all	1.0	0.75	0.87	0.83	**1.1**
datastar/express	1.0	**1.1**	0.95	1.0	1.0
datastar/high	1.0	0.75	1.1	1.1	**1.2**
datastar/normal	1.0	0.82	1.0	**1.2**	1.1
datastar/TGnormal	1.0	1.0	1.1	**1.3**	**1.3**
llnl/all	1.0	0.79	**1.1**	1.0	**1.1**
lonestar/normal	1.0	0.65	**1.3**	1.0	1.0
lonestar/serial	1.0	1.0	1.0	**1.1**	**1.1**
sdscblue/express*	**1.0**	0.80	**1.0**	0.92	0.74
sdscblue/high	1.0	0.88	0.91	**1.2**	1.1
sdscblue/low	1.0	1.0	1.1	1.1	**1.2**
sdscblue/normal	1.0	0.80	1.1	**1.3**	1.2
sdsc/express	1.0	**1.1**	0.97	1.0	0.97
sdsc/high	1.0	0.96	**1.7**	1.3	1.3
sdsc/low	1.0	1.2	**1.3**	1.2	1.2
sdsc/normal	1.0	0.90	1.1	1.2	**1.3**
ucteragrid/dque	1.0	0.92	**1.1**	1.0	0.96

Table 3: The ratio of "plain" BMBP's RMS error to that generated by the application of static and automatic clustering using each variable.

Using the 0.95 quantile, over time we would expect that at least 95% of the measurements would be less than their predicted bounds, if the methodologies are rigorous. Notice that occasionally, the correctness percentage achieved by some method is only 94% (denoted in italic font in the table). Both BMBP with and BMBP without clustering are time-series techniques that attempt to account for correlated response and change-points. While the overall percentage should be greater than or equal to 95%, there can be periods of time (particularly after a change-point) where each method experiences a number of closely correlated failed predictions. In Table 2 we show the correctness percentage that occurred at the end of each simulation. Because each method maintains a percentage close to 95%, a change-point near the end of the trace causes the overall percentage to dip below 95 without enough remaining time to recover. That is, throughout each trace, there are brief periods following a change point where the correctness percentage falls below 95%. In the cases where the trace ends before the predictors can recover, the reading we obtain is 94%. We provide a more detailed analysis of BMBP's correctness and its response to change points in [3]. Here, we note that this data indicates the introduction of clustering does not break the rigor of BMBP.

Notice also that BMBP with clustering should be far more sensitive to this phenomenon than "plain" BMBP since it is routinely making predictions that are one to two orders or magnitude lower (as depicted in Figures 1 and 2) and hence is more likely to experience a greater number of under predictions. We discuss this phenomenon more extensively in Subsection 4.5 and note here that the clustering algorithm were miscategorizing jobs so that a large number of small predictions were made for jobs that actually experienced large wait times, we would expect to see correctness percentages well below 95% in those cases. Thus the results in Table 2 also confirms that the clustering algorithm is not mis-clustering the data.

These results also contain evidence that explains the effectiveness of clustering. In the Lonestar *serial* queue, only single-processor (sequential) jobs may execute, thus, requested run time is the only factor that can affect scheduling priority (again, presumably due to the use of backfilling, which we confirmed is employed in some measure at TACC). Similarly, we confirmed that SDSC has been using some form of backfilling in the *normal* queue on all three of the machines we studied (Datastar, the SDSC SP-2 and the SDSC Blue Horizon). Again, requested execution time yields either the best ratio or almost the best in these cases. On the other hand, we cannot offer an explanation for the lack of effect in the *express* queues. Maximum requested execution time and node count are severely limited in these queues so it may be that the method over-clustered the relatively homogeneous data. In addition, not all users have access to the *express* queue at all times, which may also introduce a similarity between jobs that the clustering algorithm cannot detect.

Somewhat surprisingly, the administrator-determined static partitioning is out-performed by the automatic clustering method in all cases but two (the SDSC Blue Horizon and Datastar *express* queues). We note that our method (which adjusts its clustering dynamically) often achieves a more accurate result than the static approach; we suspect that user demands vary too dynamically for a static partitioning to be effective.

4.5. Discussion

Table 2 shows that both "plain" BMBP and BMBP with clustering occasionally fail to achieve the target correctness percentage. Both methods are susceptible to upward trends. To be useful as a method of providing soft guarantees on scheduling delay, our new method must remain correct in the face highly variable queues. Given the variability in delay, the method must be able to detect and reconfigure its internal prediction state fast enough to maintain its statistical bounds.

We illustrate this feature by showing (in Figure 3) a 12-day period (June 22nd through July 4th 2005) from the Datastar *high* queue during which time such a trend in delay is evident.

The delay experienced by job #43 in this trace (submitted at 9:56 AM on June 24th) was 15,840 seconds (7 hours and 19 minutes). Job #44, submitted at 11:55 AM, experienced a queuing delay of 217,878 seconds (60 hours and 31 minutes). From that point forward until job #184, submitted on June 27th at 11:31 AM, the queuing delay increases monotonically for each successive job, peaking at 1,763,936 seconds (20 days, 8 hours, 59 minutes). Thus, during a three-day period for this queue, the delay experienced by submitted jobs climbs steadily through three orders of magnitude. Note that the x-axis of the graph shows job sequence number rather than the time of submission. If it did, the monotonic trend would appear as an almost vertical "spike"; such spikes occur on four separate occasions in the approximately 1.5 year trace for this queue.

Despite the extreme variability of wait times in this queue, BMBP with clustering is able to handle this case in two ways. First, it detects change-points relatively

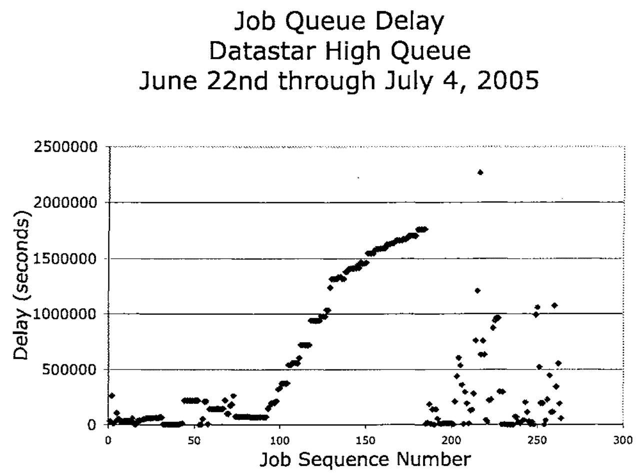

Fig. 3. Queue delay measured in seconds, all jobs, Datastar *high* queue, June 22nd through July 4th 2005

rapidly with respect to the durations associated with the trends. Secondly, it automatically identifies only a few job categories (in this case only three). When a spike occurs, it immediately trims the history it uses and begins using the largest of its categories to make predictions (thereby defaulting to "plain" BMBP) until steady-state returns.

We believe this example illustrates the utility of both approaches (used in combination) as a diagnostic tool for the user or administrator. In Figure 4 we show the 0.95 BMBP predictions without clustering for both the Datastar *high* queue and the *normal* queue from July 3rd through July 5th.

Starting late on the 3rd, the predictions for the *high* queue start to increase as BMBP recognizes a series of consecutive misses as a change-point. By 8:30 AM on the 4th, the predictions exceed those for the *normal* queue. As a user or administrator of this system, such an effect is almost surely cause for action. An administrator might react to this event by adjusting the queue priority policies or making sure the software has not experienced a failure, while a user could simply decide to submit their jobs to the normal queue until the benefit of submitting to the high queue (and paying more) becomes once more apparent.

We note that the way in which we have chosen to measure accuracy is somewhat biased against the clustering enhanced version of BMBP. Since "plain" BMBP mis-predicts jobs with relatively long delays, it disproportionately mis-predicts those jobs that would fall in the clusters corresponding to longer wait times. If, as we posit, these jobs also have the largest resource requirements, it is likely that the mis-predictions are for users who are most interested in accurate bounds (i.e. those with the largest resource requirements). On the other hand, BMBP with clustering attempts to "spread" the mis-predictions among various clusters evenly. As a result, even in the cases where it achieves a higher RMS error overall for some particular

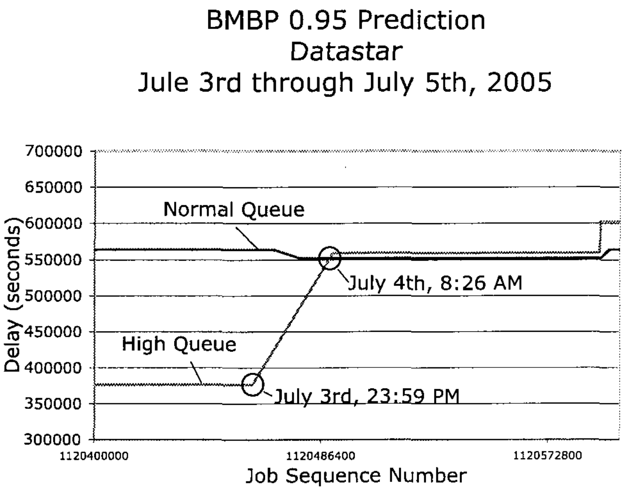

Fig. 4. BMBP 0.95 predictions without clustering, all jobs, Datastar *normal* and *high* queues, July 3^{nd} through July 5^{th} 2005

response variable, users may find the new method's results more useful.

Finally, the performance of BMBP with clustering is quite good. Using a 1-gigahertz Pentium III, the average time required to make each prediction over all batch queue logs is approximately 8 milliseconds. With this performance cost, it is possible to run our method on-line, in direct response to user queries. The current alpha release distribution of our tools (which also form the computational engine behind the web site) use only BMBP and the static node partitionings suggested to us by each site administrator. The software current uses the Network Weather Service (NWS) [28,27,26] – an open-source distributed performance monitoring and forecasting service – to gather and manage job submission data. Our intention is to release BMBP with clustering as NWS forecasting components (the NWS currently includes a number of on-line statistical prediction facilities). Presently, however, only the monitoring components are part of the production NWS release. Indeed, the motivation for this study is, in part, to verify the efficacy of BMBP with clustering prior to committing the engineering effort necessary for its support as part of the public NWS distribution.

5. 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 [3] 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 Batch Predictor (BMBP) – outperforms competitive approaches.

Site administrators, however, use scheduling policies such as backfilling, as well as various explicit priority schemes, to balance the need for high resource utilization with the desire to minimize the queuing delay experienced by users. Many of these policies use requested execution time, requested node count, or the product of the two to determine how a job will be prioritized. Thus, taking these characteristics into account as a way of categorizing jobs should yield tighter predictions of queue delay bounds.

We therefore introduce an enhanced BMBP, which uses a hierarchical clustering scheme, to improve the accuracy of the bounds generated by "plain" BMBP, a non-parametric approach to estimating bounds for batch-queue wait times. We find that using our method to cluster jobs according to their requested time generally results in better predictions than those generated by BMBP alone and is more efficacious than clustering either by requested number of nodes or by requested number of node-seconds.

In the future, we will consider other approaches to the problem of clustering. No clustering method short of exhaustive search is guaranteed to find an optimal clustering (in terms of, in our case, BIC); it may be that for data of the type we are considering, we are better off using a partitional approach or model-based k-means. In addition, the sensitivity of clustering to chosen model should be investigated: it may be that by choosing a family of models that are more accurate than the exponential at the cost of some extra complexity (*e.g.*, Weibull distributions) will produce clusters for which BMBP can perform better. In addition, it is possible to employ a multidimensional clustering scheme, so that we can evaluate the effectiveness of using both requested time and requested nodes as clustering variables. There are also other variables (queue length, longest current wait time in the queue, requested processor type, requested memory, *etc.*) that might also be used as explanatory variables for wait times and could therefore be used as a basis for clustering. Finally, we plan to monitor the effects of clustering enhanced BMBP deployment for scientific user communities to discern its ultimate effects on workload and queuing performance.

Acknowledgements

This work was supported by grants from the National Science Foundation numbered CCF-0331654 and NGS-0305390.

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. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of ACM Symposium on the Principles and Practice of Parallel Programming (PPoPP) 2006*, March 2006.

[4] Su-Hui Chiang and Mary K. Vernon. *Dynamic vs. Static Quantum-based Processor Allocation.* Springer-Verlag, 1996.

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

[6] Allen Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.
[7] Allen Downey. Using queue time predictions for processor allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.
[8] The Dror Feitelson's Parallel Workload Page. `http://www.cs.huji.ac.il/labs/parallel/workload`.
[9] Dror G. Feitelson and B. Nitzberg. *Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*. Springer-Verlag, 1996.
[10] Dror G. Feitelson and Larry Rudolph. *Parallel Job Scheduling: Issues and Approaches*. Springer-Verlag, 1995.
[11] Dror G. Feitelson and Larry Rudolph. *Towards Convergence in Job Schedulers for Parallel Supercomputers*. Springer-Verlag, 1996.
[12] Eitan Frachtenberg, Dror G. Feitelson, Juan Fernandez, and Fabrizio Petrini. *Parallel Job Scheduling Under Dynamic Workloads*. Springer-Verlag, 2003.
[13] C.W.P. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.
[14] Gridengine home page – `http://gridengine.sunsource.net/`.
[15] Mor 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.
[16] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
[17] Dave Lifka. *The ANL/IBM SP scheduling system*, volume 949. Springer-Verlag, 1995.
[18] David Lifka, Mark Henderson, and Karen Rayl. Users guide to the argonne SP scheduling system. Technical Report TM-201, Argonne National Laboratory, Mathematics and Computer Science Division, May 1995.
[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. pages 281–297, 1967.
[20] Maui scheduler home page – `http://www.clusterresources.com/products/maui/`.
[21] Cray NQE User's Guide – `http://docs.cray.com/books/2148_3.3/html-2148_3.3`.
[22] Pbspro home page – `http://www.altair.com/software/pbspro.htm`.
[23] Christian Posse. Hierarchical model-based clustering for large datasets. *Journal of Computational and Graphical Statistics*, 10(3):464–??, 2001.
[24] G. Schwartz. Estimating the dimension of a model. In *Ann. of Statistics*, pages 461–464, 1979.
[25] Warren Smith, Valerie E. Taylor, and Ian 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.
[26] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, January 1998.
[27] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, March 2003.
[28] R. Wolski, G. Obertelli, M. Allen, D. Nurmi, and J. Brevik. Predicting grid resource performance on-line. In A. Zomaya, editor, *(Handbook of Innovative Computing: In-*

tegrating Classical Models with Emerging Technologies. Springer-Verlag, Fall 2005.
[29] Shi Zhong Zhong. Journal of machine learning research 4 (2003) 1001-1037 submitted 3/03; revised 7/03; published 11/03 a unified framework for model-based clustering.