CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **28**:2685–2710
Published online 23 December 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3735

Qespera: an adaptive framework for prediction of queue waiting times in supercomputer systems

Prakash Murali[1] and Sathish Vadhiyar[2,*,†]

[1]*IBM Research, Bangalore, India*
[2]*Supercomputer Education and Research Center, Indian Institute of Science, Bangalore, India*

SUMMARY

Production parallel systems are space-shared, and resource allocation on such systems is usually performed using a batch queue scheduler. Jobs submitted to the batch queue experience a variable delay before the requested resources are granted. Predicting this delay can assist users in planning experiment time-frames and choosing sites with less turnaround times and can also help meta-schedulers make scheduling decisions. In this paper, we present an integrated adaptive framework, Qespera, for prediction of queue waiting times on parallel systems. We propose a novel algorithm based on spatial clustering for predictions using history of job submissions and executions. The framework uses adaptive set of strategies for choosing either distributions or summary of features to represent the system state and to compare with history jobs, varying the weights associated with the features for each job prediction, and selecting a particular algorithm dynamically for performing the prediction depending on the characteristics of the target and history jobs. Our experiments with real workload traces from different production systems demonstrate up to 22% reduction in average absolute error and up to 56% reduction in percentage prediction error over existing techniques. We also report prediction errors of less than 1 h for a majority of the jobs. Copyright © 2015 John Wiley & Sons, Ltd.

Received 6 August 2015; Revised 6 November 2015; Accepted 11 November 2015

KEY WORDS: queue waiting time; batch queue; scheduling in supercomputers

1. INTRODUCTION

Traditionally, large compute systems offer massively parallel operations using a large set of processors, specialized interconnects, and appropriate resource management software.The design and implementation of suitable resource management middleware are crucial to enable productive research usage of these systems. In High Performance Computing (HPC) systems, the high system acquisition cost mandates the deployment of system software that imposes minimum overheads on user applications and scheduling algorithms that can maximize the system utilization. Most production supercomputers allow users to submit jobs through batch submission queues and employ scheduling policies that use space sharing of the available processors to partition resources among multiple users or applications. Well-known parallel job management frameworks like IBM LoadLeveler [1], portable batch system (PBS) [2], platform load-sharing facility (LSF) [3], and Maui scheduler [4] are used to provide resource management services for system administrators and job queuing and execution services for users on these supercomputers. These frameworks allow system administrators to use policies like backfilling, which increase utilization by modifying the First Come First Serve (FCFS) queuing order by allowing a job to start before previously submitted jobs. Policies like fair share, user and queue prioritization, and limits on the number of queued and running jobs are also implemented to allow equitable access to resources.

*Correspondence to: Sathish Vadhiyar, Supercomputer Education and Research Center, Indian Institute of Science, Bangalore, India.
†E-mail: vss@serc.iisc.ernet.in

Copyright © 2015 John Wiley & Sons, Ltd.

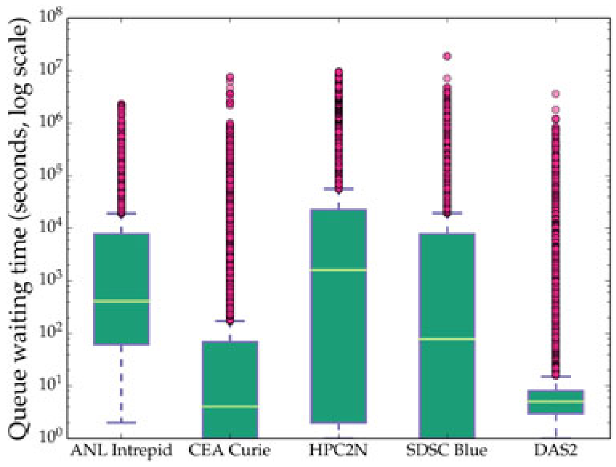


Figure 1. Queue waiting times seen in production HPC systems.

On space-sharing systems, it is expected that users request a set of compute nodes for a particular duration of time. With multiple users contending for the compute resources, a batch queue submission incurs time because of waiting in the queue before the resources necessary for its execution are allocated. The queue waiting time ranges from a few seconds to even a few days on most production systems. Figure 1 shows the box plot of queue waiting times observed in the workloads at five production systems featured in parallel workloads archive (PWA) [5]. In each box, the lower and upper edges denote the lower and upper quartile of the set of queue waiting times and the middle line shows the median. The whiskers mark the interquartile deviation (1.5×) from the median. From the large number of outlier points with high waiting times and the small values of lower quartile and median, we can see that queue waiting times vary widely within and across systems.

Prediction of queue waiting times for jobs enables users to estimate job turnaround time, which can be used for planning and task management. Additionally, reliable predictions will allow the users to tune job parameters like requested number of nodes or estimated running time to achieve faster response times. Such predictions can also be efficiently used by a meta-scheduler to make automatic scheduling decisions for selecting the appropriate number of processors and queues for job execution. Usage statistics of predictors like queue bounds estimation from time series (QBETS) [6] that was used to provide bounds on the queue waiting times in TeraGrid [7] and the Karnak Prediction Service [8], which is currently used in seven Extreme Science and Engineering Discovery Environment [9] supercomputers, indicate that users are keen to incorporate predictions for managing experiment time lines and to efficiently schedule complex scientific workflows across multiple supercomputing sites [10].

Developing a site-independent prediction system that supplies the user a real valued waiting time prediction at the instant of job submission is challenging because of a complex interplay between scheduling algorithms, queue and user-specific usage policies, node failures, unpredictable job arrival patterns, and queue churn induced by job cancellations. Scheduling algorithms that use backfilling or policies that resemble shortest job first allocation result in dynamically varying system and queue states. Coupled with irregular job arrival patterns, it becomes difficult to anticipate the state changes of the batch queue and design scheduling algorithm agnostic prediction strategies. Additionally, site-specific node partitioning constraints on architectures such as BlueGene/P make the problem harder.

In this paper, we present an integrated adaptive framework, Qespera,[‡] to predict queue waiting times for jobs submitted to production parallel systems. We propose a novel algorithm based on spatial clustering for examining the cluster structure of similar jobs in the history. Using the cluster structure, our adaptive method chooses an appropriate prediction model among a standard

[‡]Espera means wait in Spanish.

deviation minimizer (SDM), ridge regression, and a nearest neighbor (NN) predictor. Our framework uses the processor occupancy state and batch queue state along with job characteristics like requested number of nodes and user estimated running time to find similar jobs in the history.

The following are the primary contributions of our work.

- The use of distributions over feature summaries to represent queue and processor occupancy states.
- Features that consider user-specific system utilization/demand policies that are prevalent in many supercomputing sites.
- A novel algorithm that uses spatial clustering to predict job waiting time based on very similar past submissions.
- An adaptive scheme that selects the most relevant features for prediction for each individual job.
- A simple characterization of the feature neighborhood of a job that is used to select the correct prediction model for each job.

We have evaluated our adaptive prediction framework using workload simulation traces from the PWA [5]. Our test set comprises production workloads with varying site and job characteristics, including two Top500 systems. Across workloads, our predictions result in up to 22% reduction in the average absolute error and up to 56% reduction in the percentage prediction errors over existing techniques. We also report prediction errors of less than 1 h for a majority of the jobs for most workloads. In addition, for one of the systems, we use hardware failure traces and site-specific scheduling policies to determine the reasons for large prediction errors for some jobs. We are able to account for the high prediction errors for most of these jobs. We have also integrated our framework with OpenPBS and deployed it on a cluster in our department.

In Section 2, we present existing prediction algorithms. In Section 3, the proposed framework is described in detail, including the job and system features used, calculation of distances between the jobs, and prediction models used for obtaining predicted queue waiting times. Section 4 describes the experimental setup and evaluation results for various supercomputer job traces. In this section, we present comparison results with two previously proposed wait time prediction strategies. In Section 5, we share our experience of deploying our framework in a cluster in our department. Section 6 discusses certain aspects that impact the predictions. Section 7 includes a summary of the paper and plans for future work.

2. RELATED WORK

Earlier efforts on prediction of queue waiting times can be broadly classified into non-statistical and statistical methods. Non-statistical methods are based on exact simulation of the scheduling algorithm and decisions that would be made by the scheduler in real time. This approach inherently limits the applicability of these techniques because scheduling policies are hard to model and complete information about the scheduler is usually not published. In contrast, statistical methods do not assume specific scheduling algorithms or run time models.

2.1. *Non-statistical methods*

Smith *et al*. [11, 12] propose methods to infer job run times using similar previous submissions, and these estimates are used to simulate the scheduling algorithms including FCFS Serve, Least Work First, and backfilling [13] to estimate queue waiting times. The work by Downey [14] proposed mean and median-based predictors that assume a log uniform model of job execution times to estimate the waiting time of the job at the head of an FCFS queue. This approach cannot be used to obtain an estimated queue waiting time at the time of job submission. Hence, this approach is not suitable for metascheduling using queue waiting and response time predictions. In contrast, our proposed framework provides a queue waiting time prediction at the instant of job submission. It is not restricted to a specific scheduling policy and can be deployed across platforms with different resource allocation policies. Some of these previous efforts also use run time estimates for prediction of queue waiting times. Prediction errors for run time can propagate to yield large errors in the waiting time predictions.

Some supercomputer systems utilize discrete-event simulators that simulate the exact resource management policies employed in the system. An example is the QSIM simulator framework employed in the Argonne National Laboratory (ANL) to simulate Cobalt [15] resource management strategies. Such simulators are used for analyzing various what-if scenarios including adjusting user run time estimates [16] and can also be used to estimate queue waiting times on the particular system. Such estimates are also used to build robust job schedulers [17].

2.2. *Statistical methods*

The QBETS [6] is a forecasting system that uses quantile statistics of the history of job submissions to provide bounds on the queue wait times with a quantitative confidence level. QBETS uses a predictor based on non-parametric inference, an automated change-point detector to trim history, a hierarchical clustering algorithm to identify jobs having similar characteristics, and queue length-based downtime detection algorithm to identify system failures that affect job queuing delay. Using these components, QBETS can handle varying workload characteristics and customized queuing policies. However, QBETS gives conservative upper-bound predictions, which leads to large prediction errors for a majority of jobs. A primary limitation of QBETS is that it does not consider the state of the system and uses only the job characteristics, which we show is insufficient for efficient predictions of queue wait times.

The work by Smith [18] uses a weighted heterogeneous Euclidean-overlap distance metric to compare job attributes and find similarities between the target job and history jobs. Kernel weighted average (WA) is used to obtain a prediction from history submissions, and a genetic algorithm is used to tune the weights for the distance function. The instance-based learning (IBL) method by Li *et al.* [19–21] considers both the job characteristics and the system states for the prediction of queue waiting times. This work considers more features than the prediction methods of Smith [18] and includes other predictors like *1*-NN, the *n*-WA of *n*-NN, and locally weighted linear regression. These works also use a genetic algorithm to obtain a static weight vector, which is used to improve the similarity computations. We differ from this work on many aspects including an improved representation of the system state using distributions and user-specific job submission policies, a problem-specific prediction model, use of clustering techniques to give importance to similar queue waiting times in the history jobs, a dynamically varying weighting scheme to handle local trends in the workload, and an adaptive method to vary the prediction model for each job. We show that our method gives improved queue waiting time predictions over QBETS and IBL.

A recent work by Kumar and Vadhiyar [22] defines boundaries in the history of prior job submissions for predictions. These boundaries denote an estimate of the closeness of the history system state to the current state and allow them to classify the history jobs as near-term, mid-term, and far-term. Using these history job classes, they define a set of criteria based on the current queue and system state, which can be used to infer a bound on the waiting time. However, their method is limited to identifying whether a job belongs to the class of jobs with waiting time less than 1 h.

3. METHODOLOGY

In this section, we provide a complete description of the algorithm used to predict queue waiting times in parallel systems. We present an overall view of the algorithm, followed by a description of each component. A fundamental assumption in our method is that similar jobs that arrive during a similar system queue and processor states experience similar queue wait times. When a new job arrives in the job queue, the job scheduler takes a decision based on the arriving job's attributes and the current system state. Given a set of jobs and an appropriate definition of job similarity, a deterministic scheduling algorithm is expected take the same actions for jobs with similar resource requirements. Before such jobs start execution, they will experience a series of similar system changes if their queue and processor states at the time of arrival are approximately the same. Hence, we expect that such jobs will have similar queue waiting times.

We use two kinds of statistics with respect to features that describe processor and queue states: feature summary and distributions. We found that using distributions helps in capturing the similarities of jobs better than using feature summary if the distributions are not uniform. Using a training set, one of these two statistics is chosen. These statistics are used along with the job attributes to characterize a job at the time of its submission. We then use a weighted distance metric to calculate the similarities of the target job with the history jobs. We follow an online learning approach that uses a clustering algorithm to quickly characterize the feature neighborhood of the target job based on the distances from the history jobs. Depending on the cluster structure, we use one of three methods to calculate the predicted queue waiting time of the target job: an SDM, NN method, and ridge regression.

We describe the job features in Section 3.1 and the distance function in Section 3.2. The criteria used to analyze the cluster structure of the feature neighborhood and the prediction models are presented in Section 3.3.

3.1. Job features

At the time of arrival of a job in a supercomputer queue, certain jobs will be running on the nodes of the system and certain other jobs will be waiting in the queue. The processor state of the system contains information about the running jobs, and the queue state contains information about the waiting jobs. To predict the wait time of the new job, we look for jobs in the past that had similar resource requirements as the current job and whose processor and queue states were similar. In order to establish a working definition of job similarity, we quantify the processor and queue states associated with a job. In this section, we motivate and define a set of features, which are used in Section 3.2 to define job similarity. In the remainder of this chapter, queue and processor states are jointly referred to as system state.

Given a job j, we denote the submission time of the job in the queue by $t_s(j)$, the number of nodes/cores requested by the job by $req\_size(j)$, the estimated wall clock time of the job provided by the user by $ert(j)$ (estimated run time (ERT)), and the unique id of the user submitting the job by $user(j)$. These job attributes can be gathered from the job submission script provided by the user and are also maintained in workload logs [5].

We represent the system resource states using two types of statistics: distributions and feature summary. A job feature is represented using either a numeric or nominal attribute. Distributions are sets of quantities associated with a particular feature. For example, the set of requests sizes of jobs waiting in the queue can be represented using a histogram distribution. Feature summary, as the name suggests, summarizes the distributions to produce a representative real value. For example, a feature summary for the set of requests sizes of the waiting jobs in the queue can be the sum or average of the request sizes. Numeric attributes are real values that can be ordered using an appropriate distance function. Nominal attributes are quantities that cannot be ordered and can only be used for identification. To predict queue wait times for jobs at a supercomputing site, we use the job attributes in conjunction with either distributions or feature summary for the system state.

One of the important contributions of our work is the use of distributions over feature summary in some cases to represent and compare system states. The process of summarization loses information, which can, in some cases, be vital for correct comparisons. In Figure 2, the two processor states of a 64-node machine are indistinguishable if we consider the sum of request sizes or the number of jobs; however, the distributions of request sizes are clearly different. These two system states are different

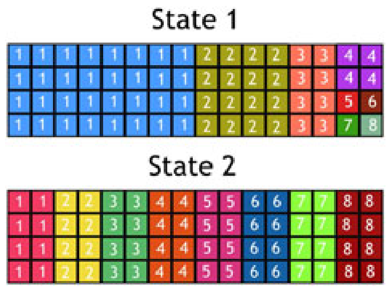


Figure 2. Two different processor states for a 64-node machine (represented as a 4 × 16 matrix). Each small square represents a node and each color represents a job.

from the perspective of job wait time because they will undergo drastically different sequences of state changes. In certain workloads, where distributions can reveal a distinct bias or skewness for some features, we found that the use of distributions can greatly improve the similarity computation and lead to better predictions. We use six distributions to represent the system state at the instant of arrival of a job: three distributions to represent the characteristics of the waiting jobs and three distributions to represent the characteristics of the running jobs. Together, these distributions capture the trends in resource requirements of queued jobs, the current load on the system resources, and some temporal characteristics of the system state. The distributions used by our predictor are listed in Table I. Because the explanatory variables of interest in the system state are all discrete quantities (time is measured in seconds), it is natural to represent each distribution using a histogram. For example, for obtaining the queue state distribution of request sizes for a particular job submission, all the waiting jobs at the time of arrival of the job are selected and the histogram of their request sizes is computed.

We chose these six features because they have direct impact on the queue waiting times and can be obtained or deduced from the job traces. For example, if the ERT of a job is high, it can expected that it will wait longer in the queue before it obtains the resources necessary for its execution. At the instant of a job submission, if the system is heavily occupied, as reflected in a high value for the sum of sizes of the executing jobs, or if the queue is populated by a significant number of large jobs, as reflected in a high value for the sum of sizes of queued jobs, we can expect a larger queue waiting time for the currently submitted job. Similar arguments are applicable to the other features in our set. In this work, we have not considered job priority as a feature because they are not available for all job traces.

In certain cases when the queues have jobs with a large number of different request sizes and ERTs, the distributions described earlier can resemble the uniform distribution or lack significant differences in kurtosis or skew, which can be used to obtain a natural ordering. For example, Figure 3 shows four different distributions, D_1–D_4, of request sizes of running jobs. It is clear that the system state represented by D_2 has correspond to higher load than the one represented by D_1. It is also reasonable to expect that a 128-node job submitted at an instant corresponding to the system state D_1 is likely to incur less queue waiting time than when it is submitted at state D_2. However, such differences are not apparent in cases of D_3 and D_4, which strongly resemble the uniform distribution. Thus, better predictions can be obtained by using distributions for system states, D_1 and D_2, and feature summaries for system states, D_3 and D_4.

In general, when the histograms do not have significant distinctness in shape, we cannot rely on them to obtain a meaningful quantification of similarity. For such cases, summaries of features are employed to check resource state similarity. The feature summaries and job attributes used by our predictor are listed in Table II. In the table, $1\{\textit{condition}\}$ denotes the indicator function, which is 1 when *condition* is true and 0 otherwise. The first two features are the job attributes that directly influence the wait time of the job. For example, if a job requests a higher number of cores, we expect it to wait for a longer period than a similar job that requires a smaller number of cores. Similarly, if a user specifies a high value of ERT, the chances of the job getting backfilled are lesser compared with a similar job with small ERT. The request size of a job is considered as a nominal attribute, and the ERT is a numeric scalar attribute. The analysis of workloads of supercomputing sites shows that request sizes tend to be a power of two and that the total number of request sizes used is less. This implies that for most request sizes, there are a large number of jobs in the workload. Computing the

Table I. Distributions used in distance computation.

No.	Type	Distribution name
1	Queue	Request sizes of queued jobs
2	Queue	Estimated run time of queued jobs
3	Queue	Elapsed wait time of queued jobs
4	Processor	Request sizes of executing jobs
5	Processor	Estimated run time of executing jobs
6	Processor	Elapsed run time of executing jobs

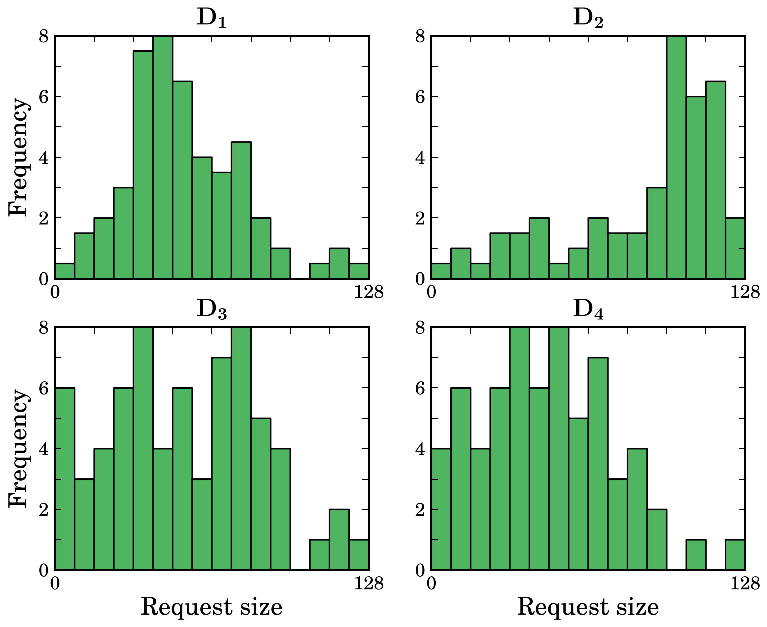


Figure 3. Example distributions.

plain difference of request sizes can make the overall distance function biased towards history jobs with lower request sizes, especially when the request sizes are power-of-two. We use 0/1 distance to ensure that a majority of the close neighbors have the same request sizes as the target job. When such neighbors are not available, the close neighborhood of the job will be sparse. We present methods to handle such feature neighborhoods in Sections 3.3.3 and 3.3.4.

Features 3–8 are the summarizations of the distributions listed in Table I. One of the unique aspects of our work is that we consider user-based queue and processor features (features 9–16). These features are intended to include site-specific policies that limit certain user-based demands or utilization. During the course of our research, we found that certain sites impose limits on the demands a user can make at a given time to ensure fairness for all users and to prevent a large number of serial job submissions by the user. For example, the system administrators of the SP2 machine at the Cornell Theory Center (CTC) used a policy that allowed only the first 6000 node hours of jobs submitted by a user to move up in the queue. Any requests from a user who has already submitted jobs requiring a total of at least 6000 node hours of CPU time are blocked at the end of the queue until one of the earlier jobs starts running [23]. We extend this idea to include policies based on other common metrics including the number of submitted and running jobs by a particular user.

3.2. Distance computation

Using the features defined in the previous section, a distance function can be used to assign a real valued similarity score for a pair of jobs. Smaller values of distance indicate higher similarity.

3.2.1. Distribution-based job distance. χ^2 (pronounced 'chi-square') distance metric is used to order the set of distributions of history jobs according to their similarity to the target job's distributions. For two histograms P and Q with K bins, the χ^2 distance is defined as

$$\chi^2(P,Q) = \sum_{i=1}^{K} \frac{(P[i]-Q[i])^2}{P[i]+Q[i]} \tag{1}$$

For each bin, the summation of bin counts in the denominator of Equation (1) implies that χ^2 distance considers small differences between large bins to be less important than a similar difference between small bins. Before applying the distance metric, each histogram bin is normalized by the

Table II. Features used in distance computation.

Jobs considered for computation	Feature	Computation
Target job (1–2)	User given request size	$req\_size(J)$
	User given ERT	$ert(J)$
All waiting jobs (3–5)	Sum of request sizes	$\sum_{i \epsilon Q} ert(i)$
	Sum of ERTs	$\sum_{i \epsilon Q} ert(i)$
	Sum of elapsed waiting times	$\sum_{i \epsilon Q}(t_s(J) - t_s(i))$
All running jobs (6–8)	Sum of request sizes	$\sum_{i \epsilon R} req\_size(i)$
	Sum of ERTs	$\sum_{i \epsilon R} ert(i)$
	Sum of elapsed running times	$\sum_{i \epsilon R}(t_s(J) - (t_s(i) + wait\_time(i)))$
Waiting jobs of the user who submitted the target job (9-12)	Sum of requested CPU time	$\sum_{i \epsilon Q} 1\{user(i) = user(J)\}req\_size(i) * ert(i)$
	Sum of request sizes	$\sum_{i \epsilon Q} 1\{user(i) = user(J)\}req\_size(i)$
	Sum of ERTs	$\sum_{i \epsilon Q} 1\{user(i) = user(J)\}ert(i)$
	Number of jobs	$\sum_{i \epsilon Q} 1\{user(i) = user(J)\}$
Running jobs of the user who submitted the target job (13-16)	Sum of requested CPU time	$\sum_{i \epsilon R} 1\{user(i) = user(J)\}req\_size(i) * ert(i)$
	Sum of request sizes	$\sum_{i \epsilon R} 1\{user(i) = user(J)\}req\_size(i)$
	Sum of ERTs	$\sum_{i \epsilon R} 1\{user(i) = user(J)\}ert(i)$
	Number of jobs	$\sum_{i \epsilon R} 1\{user(i) = user(J)\}$

ERT, estimated run time.

total frequency, which is same as the number of jobs involved in the histogram computation. This allows us to compare histograms of different queue and processor states, although the number of jobs in each histogram may be different. Once the χ^2 distance between a histogram of the target job and the corresponding histogram of a history job is determined for each of the six distributions shown in Table I, the maximum of the six distances obtained is calculated. This is used to normalize the distances, so that each pair of histograms has a distance in the [0, 1] range.

To obtain valid results using the χ^2 metric, the pair of histograms must satisfy two conditions, namely, the number of bins of both the histograms, and their bin start positions must be the same. If the training set used for prediction is used to compute maximum and minimum bounds for the histograms, the range of values of the quantities will lead to a large number of sparse bins in the histograms. For example, the request sizes can vary from a few nodes to many hundred thousands and elapsed wait times and ERTs can vary from a few hundreds seconds to hours or days. This will render the histograms unusable because no clear pattern can be discerned if most of the bins have only a few or no samples. The bounds will have to be recomputed for each new job submission whenever the feature values of a job lie beyond the bounds computed using the training set.

We employ an adaptive procedure that uses a fixed number of bins and recomputes the minimum and maximum bounds for each histogram type for each pair of jobs. For a pair of histograms, the minimum/maximum bin start/end position is taken as the minimum/maximum value of the quantity being binned among both the histograms. In this case, when the range of values is small, the bin width is small, and we can obtain a more fine-grained comparison for the histograms, and when the range of values is high, we can obtain a coarse-grained comparison of the histograms. We experimented with different number of bins and found that using ten bins gave good bin widths and results in most cases. Hence, in all our experiments, we used ten bins for each histogram.

Using the distribution distances, the final distance value for each history job is computed by adding the distances of the job attributes and applying suitable weights. Because request size is considered as a nominal attribute, $0/1$ distance is used to test whether the request sizes of the history and target jobs are the same. For ERT, plain difference with suitable normalization is used. This ensures that the distance value for each feature lies in a [0, 1] range. For a target job J and a history job h, the distribution-based job distance is defined as

$$\begin{aligned} d(J,h) =& (W(\mathit{feature}, f_1) * 1\{req(J) \neq req(j)\} \\ &+ W(\mathit{feature}, f_2) * \frac{|ert(J) - ert(j)|}{max\_ert - min\_ert} \\ &+ \sum_{i=1}^{6} W(distr, d_i) * \chi^2(D[J][i], D[h][i]))/W_{sum} \end{aligned} \tag{2}$$

where W is the weighting scheme defined in Section 3.2.3. $W(\mathit{feature}, f_i)$ and $W(distr, d_i)$ are the weights of the i^{th} feature and distribution, respectively. W_{sum} denotes the sum of weights used for the different features. *max_ert* and *min_ert* are the maximum and minimum of the ERTs, respectively, seen among the jobs in the history set and target job. $D[h][i]$ and $D[J][i]$ are the i^{th} histograms of jobs J and h, respectively. The overall distance value is in the [0, 1] range because the individual distances used in the WA are in the [0, 1] range.

3.2.2. Feature summary-based job distance. For computing the feature summary-based distance between a target and history job, 0/1 distance of request sizes and normalized plain differences of other feature values are averaged with suitable weights obtained using the weight function in Section 3.2.3. For a target job J and a history job h, the feature summary-based distance is defined as

$$d(J,h) = d_n(J,h)/W_{sum} \tag{3}$$

$$\begin{aligned} d_n(J,h) =& W(\mathit{feature}, f_1) * 1\{req(J) \neq req(h)\} \\ &+ \sum_{i=2}^{16} W(\mathit{feature}, f_i) * \frac{|F[J][i] - F[h][i]|}{max_{f_i} - min_{f_i}} \end{aligned} \tag{4}$$

In the aforementioned equations, $F[k][i]$ denotes the value of the i^{th} feature of job k and max_{f_i} and min_{f_i} are the maximum and minimum values, respectively, of the i^{th} feature seen among jobs in the history set and target job.

3.2.3. Correlation-based feature weights. Assigning real valued weights to each feature allows the distance function to give more importance to relevant features. For instance, if FCFS policy is used to schedule jobs, a predictor that estimates the wait time at the arrival time of the job need not consider the ERT of the target job. When the requested number of nodes becomes available, the job at the head of the queue will start, irrespective of its requested time, and the order in which jobs move up in the queue is based only on arrival time. One scheme for deriving weights is to use the correlations between features and queue waiting times for a training set of history jobs, and use these weights for subsequent predictions. However, such a static scheme will be incapable of handling dynamic policy changes by system administrators or varying workload characteristics, which can potentially alter the relevance of the features. In order to implicitly include such changes and capture the recent dominant trends in the workload, we recompute the weights when the history changes with addition of the latest job. The recomputations of the weights will have to be light-weight because they have to be performed for each prediction. Hence, elaborate explorations, for example the genetic algorithm scheme in [19, 20] for determining the weights, cannot be performed. We use correlation computations for calculating weights. In particular, we use the absolute values of Spearman's rank correlation coefficient, ρ, as weights for different features. We chose Spearman's correlation over Pearson's correlation of the unranked variables because the ranking of variables makes Spearman's correlation less sensitive to outliers in the data. Spearman's rank correlation coefficient, ρ, of two vectors, X and Y, of lengths N is defined as

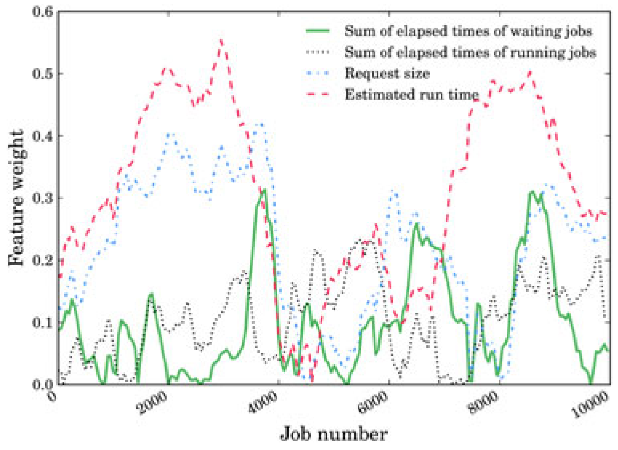


Figure 4. Change of feature weight in HPC2N.

$$\rho(X,Y) = \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2\sum_{i=1}^{N}(y_i-\bar{y})^2}} \tag{5}$$

In the case of feature summaries or job attributes like request size or ERT, the weight is the correlation of the feature value with the wait time of the job. For distribution-based features, we choose the weight as the correlation of the L^2 or vector norm of the histogram with the wait time of the job.

We observed that although the changes in the weights from a particular job to the next may be minor, such changes accrue over many jobs and over time and result in significant changes in the set of highly relevant features. For example, Figure 4 shows the change in weights or relative relevance of four features over time for the workload trace of the Seth machine at High-Performance Computing Center North (HPC2N), Sweden, obtained from Feitelson's workload archive [19, 20]. The graph clearly shows that the weights show wide variations over time and hence will have to be adapted as in our method. Comparing our weight calculation scheme with the earlier genetic algorithm approach [19, 20], we found that our computations are lightweight and take 30 ms per job, while the genetic algorithm approach takes about 20 min for evaluating one generation of solutions. We also experimentally found that the resulting predictions are equivalent to and sometimes even better than the predictions based on the genetic algorithm approach.

3.3. *Prediction models*

The steps described in the previous sections compute job similarities between a target job and the history jobs and give each history job a real valued distance value in the [0, 1] range. We developed three prediction models, namely, *standard deviation minimizer (SDM)*, *regression-based*, and *WA* methods that use the waiting times of the history jobs for predictions. In our experiments, we found that the relative merits of the prediction models for a particular target job depend on the structure of the relationships between the waiting times and the distances in the history set. We use a density-based clustering method to determine the structure of the relationships. In this section, we first describe the clustering method and then the three prediction models.

3.3.1. Density-based clustering. We observed that the relationships between the waiting times of the history jobs and the distances to the target job exhibit certain characteristics, which can be exploited to obtain good predictions. Specifically, we found three common patterns that allow us to choose the appropriate prediction model for a target job. Figure 5 illustrates these three patterns of relationships between distances (x-axis) and queue waiting times (y-axis) using a sample trace from the CTC available in Feitelson's workload archive [5]. The y-axis of the plots in the figure are adjusted so that the wait time of the target job corresponds to the line $y = 0$. Note that although the y-axis labels are unknown while predicting the wait time for the job, the structure of the graph

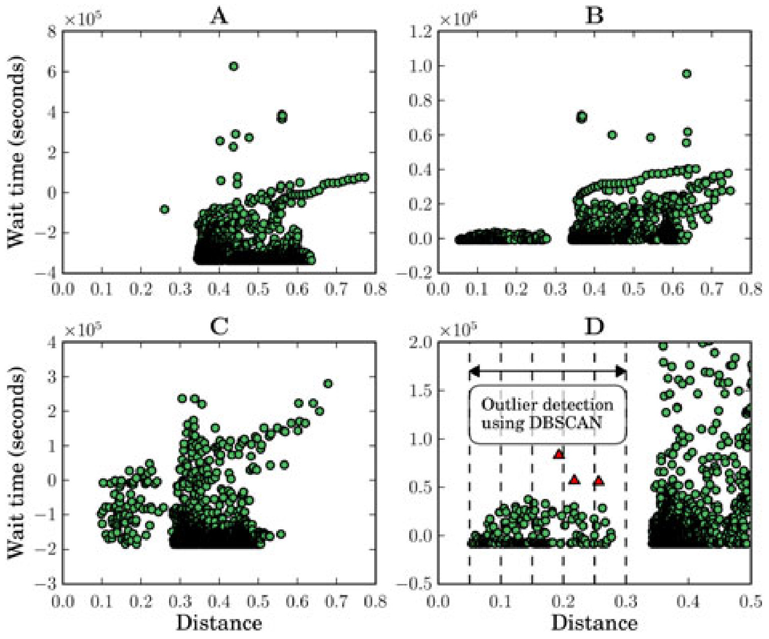


Figure 5. Typical structure of the feature neighborhoods in the Cornell Theory Center. The plots show the difference of wait time of the history job and the wait time of the target job versus the distance of the history job. Figure A exhibits the case where the history jobs are not similar to the target job. Figure B shows the case where the close neighbors form a tight cluster. In contrast, Figure C shows several close outliers. Figure D illustrates the windows used in standard deviation minimization; the red triangles are sample outliers. These jobs were handpicked from the Cornell Theory Center trace to illustrate the common patterns.

is independent of the *y*-axis labeling. Graph A shows the case where most of the jobs in the history set are far from the target job. Graph B shows the case where there is a dense clustering of jobs with small variance in wait time very close to the target job. Graph C shows the case where there are near neighbors but they have dissimilar wait times. In cases A and C, we prefer not to use the near neighbors directly for prediction because either they are too dissimilar to the target job or their wait times are dissimilar. For every target job, it is crucial to detect these patterns correctly so that the correct prediction model can be applied. We can easily distinguish case A from the other two cases by checking the average distance of the closest $k\%$ of the jobs. If the average distance is greater than a threshold, we infer that the neighbors are too far away. In order to distinguish cases B and C, we use density-based clustering.

Density-based approaches view clusters as dense collections of points separated by sparse regions of low density. We use density-based spatial clustering of applications with noise (DBSCAN), a density-based clustering algorithm that uses a linear number of range queries to grow clusters that maximize a density connectedness criterion. To define density, DBSCAN uses two parameters: ϵ and *MinPts*: *If the ϵ neighborhood of a point has at least MinPts points, the neighborhood of the point is considered dense.* Using the definition, each point in the dataset is classified as a core point, border point, or as an outlier. An important advantage of using DBSCAN over other popular algorithms including k-means clustering is that the algorithm does not require the number of clusters to be specified a priori.

Given the wait times of the history jobs, their distance values, and appropriate values of ϵ and *MinPts*, the cluster structure of the closest $k\%$ of the history jobs is examined. DBSCAN outputs a set C_k of clusters along with a set O_k of outliers. From C_k, the average size C of a cluster is determined. If the number of outliers $O = |O_k|$ satisfies the following condition, our predictor assumes that there is a good clustering structure.

$$O \leqslant f * C : f \in [0, 1] \tag{6}$$

For finding the values of k, f, ϵ, *MinPts*, and the window size, we used a small training set for tuning the parameters. Section 4 contains more details on this tuning procedure and the parameter values.

3.3.2. Standard deviation minimizer. SDM assumes that a close dense cluster of jobs exists in the neighborhood of the target job. If the cluster of jobs is dense, it implies that the jobs have comparable distance with the target job and they experienced similar queue wait times. To explain the intuition behind SDM, we consider case B in Figure 5. In this case, we can see that the cluster of jobs with distance less than 0.2 has no significant outliers (as compared with case C). We see that along the *y*-axis, the cluster is tightly centered around the $y = 0$ line. This means that the cluster's average wait time is a good approximation for the wait time of the target job. To refine this cluster further, we divide the graph into a set of distance-based windows (along the *x*-axis) and compute the standard deviation of wait times for jobs within each window. The vertical lines in the fourth graph in Figure 5 demarcate windows of length 0.05 for a job in CTC. To obtain a very accurate prediction, window of jobs that has minimum standard deviation and minimum distance to the target job has to be selected. The windows are formed only for a set of jobs that have distance less than a maximum threshold. This is because large variations in wait times are expected and observed in the farther regions of the graph where the jobs are highly dissimilar to the target job.

While computing the standard deviation of jobs in a particular window, we eliminate any outliers present in that window (triangles in Figure 5). To find the outliers, we reapply DBSCAN on the part of the graph up to which SDM looks for clusters. Recall that for determining whether we have a good cluster in the near neighborhood, we used only a small set composed of the closest $k\%$ of the jobs, where $k = 5$ in our experiments. Now, we expand the set and apply DBSCAN a second time to obtain the full set of outliers in the region of interest (0 to 0.3 in case of CTC). Among the windows within the maximum distance threshold, the one with the smallest standard deviation is selected, and the average wait time of its jobs is reported as the wait time of the target job. For computing the average of the wait time of the jobs in the cluster, we use a weighted scheme where the Gaussian kernel ($e^{d^{-2}}$, where d is the distance) is used to assign higher weights for jobs with smaller distance from the target job. From Graph B in Figure 5, it is clear that selecting the window size and distance up to which SDM looks for the tight cluster is very important to the accuracy of the algorithm. If SDM crosses the distance of 0.3 and selects a window with small standard deviation among the farther jobs, the prediction is not expected to be correct. We pick suitable values for the various SDM parameters by performing sensitivity studies using a training set.

3.3.3. Regression. When SDM cannot be applied because of poor clustering, the feature summaries of the jobs are used to construct a linear regression model. A simple least squares regression cannot be used for our purpose because we found that our dataset has the problem of multicollinearity. The input features to regression are termed multicollinear if there are strong linear relationships between the features.

In ordinary least squares regression, for a feature matrix X, matrix inversion of $X^T X$ is used to compute the solution, which minimizes a squared error function. If the matrix X is ill-conditioned, the regression coefficients can be highly inaccurate, and small changes in the data can cause significant variations in the coefficients. For example, in a supercomputer trace, obtained from Feitelson's workload archive, corresponding to the Intrepid system of the ANL, we found high correlations between the job features, with Pearson's correlation coefficients greater than 0.75 for eights pairs of features. This indicates that our dataset has multicollinearity. The correlation coefficients also indicate the degree of ill-conditioning introduced by pairwise linear relationships between the features. In fact, we found the condition numbers of the feature matrices for different traces to be high and up to 65,000, thus showing that the matrices are ill-conditioned.

We use ridge regression because it employs a quadratic regularization term to shrink the values of the regression coefficients, making them more stable and robust to collinearity. The feature vectors are normalized to have mean 0 and variance 1 before the regression is computed. To compute the wait time for a job, the model is evaluated using target job's features.

3.3.4. Weighted average. The third method we used is k-WA-based predictions, if the regression outputs a negative value for the wait time. To use weighted average, the distances calculated using feature summaries are used to assign weights to jobs as described in Section 3.3.2. A set of k-NN is then chosen, and the WA of their wait times is reported as the prediction. Based on experiments with different values of k, we choose a suitable value for each trace.

3.4. Wait time prediction system

The components described in the previous sections are used to develop a complete wait time prediction system. We present the pseudo code of the prediction system in Algorithm 1. For each trace, a small set of jobs is used to pick either distributions or summary features for the distance computation (lines 1–5). *use_distribution* is *True* if distributions are chosen, else it is *False*. The other parameters required for the algorithm are tuned using a validation set consisting of 4000 jobs (from job number 16000 to 20000 in each dataset). When the user submits a job to the batch queue, Algorithm 1 is invoked to obtain a wait time prediction. The algorithm uses the appropriate distance function to evaluate the average distance and outlier criteria (lines 6–12). If these conditions are satisfied (lines 14 and 15), standard deviation minimization is used. If not, the algorithm uses ridge regression to find the wait time of the target job (line 17). When the regression outputs a negative wait time value, the prediction from WA is reported to the user (lines 18 and 19).

Algorithm 1 QESPERA: WAIT-TIME-PREDICTOR (J, H)

```
Require: use_distribution, dist_threshold, k, f
Ensure: wait_time
 1: if use_distribution is True then
 2:     dist= DISTRIBUTION-BASED-JOB-DISTANCE (J, H)
 3: else
 4:     dist= COMPUTE-FEATURE-DISTANCE (J, H)
 5: end if
 6: sort(dist)
 7: C = closest k% of jobs from H
 8: $\overline{d_C}$ = average distance of jobs in C
 9: clusters $clust_C$, outliers $out_C$ = DBSCAN(C)
10: $\overline{s_C}$ = average size of sets in $clust_C$
11: $o_C = |out_C|$
12: wait_time = 0.0
13: if $(\overline{d_C} < dist\_threshold) \wedge (o_C \leqslant f * \overline{s_C})$ then
14:     wait_time =STANDARD-DEVIATION-MINIMIZER (D)
15: else
16:     wait_time =RIDGE-REGRESSION-PREDICTOR(J, H)
17:     if wait_time < 0 then
18:         wait_time =WEIGHTED-AVG-PREDICTOR (J, H)
19:     end if
20: end if
21: return wait_time
```

4. EXPERIMENTS AND RESULTS

4.1. Experimental setup

The experiments were conducted using real workload traces of large-scale production parallel systems available from the PWA [5]. Each trace, available in the standard workload format (SWF) [24], contains information about the chronology of job submissions, service times offered by the batch queue, and other auxiliary characteristics that are useful for workload modeling and simulation.

Table III. Supercomputer workload traces used in our evaluation.

System	Time period	Max. CPUs	Number of jobs	System utilization
SDSC Paragon	December 1994–December 1995	400	76872	71.7
CTC SP2	June 1996–May 1997	338	79302	85.2
SDSC SP2	April 1998–April 2000	128	73496	83.7
SDSC BLUE	April 2000–January 2003	1152	250440	76.8
HPC2N	July 2002–January 2006	240	527371	70.2
DAS2	January 2003–January 2004	144	225711	14.9
ANL Intrepid	January 2009–September 2009	163840	68936	59.6
CEA Curie	February 2011–October 2012	93312	773138	29.3

Table IV. Scheduling algorithms used in the considered systems.

System	Scheduling policy
SDSC Paragon	Unknown
CTC SP2	Backfilling with user based limits [23, 25]
SDSC SP2	Unknown
SDSC BLUE	Unknown
HPC2N	Priority scheduling using fair share, user priorities, and user-based limits [26]
DAS2	Backfilling [27]
ANL Intrepid	Priority scheduler with user-based limits [28]
CEA Curie	Fair share scheduling [29]

For workload simulation, a discrete-event simulator that reads the SWF trace is used to replay the job submission, start of job execution, and job termination in a chronological sequence. When a job arrives, the simulator adds the job to the list of waiting jobs in the appropriate queue. When the job begins execution on the supercomputer, the simulator moves the job from the list of queued jobs to the list of running jobs. When a job finishes running, its entry is purged from the list of running jobs. If a job fails while running, the SWF trace contains the runtime until failure, and for the purpose of simulation, the event is considered equivalent to normal job termination. However, if a job is canceled while it waits in the queue, it is simply removed from the list of waiting jobs at the appropriate time. Using these lists, the simulator can maintain an online set of history job submissions that can be used for obtaining predictions for arriving jobs. The history set is updated when a currently waiting job is removed from the waiting list. When a new job entry is added to the history, the earliest entry is removed. The simulator can also interface with standard resource managers like PBS [2], platform LSF [3], or IBM Loadleveler [1]. For instance, on PBS-based batch queues, *qstat -f* command provides the information necessary for our framework to monitor the system state.

We have selected a set of eight traces, shown in column 1 in Table III, that contain sufficient information to reconstruct the queue and processor state of the system at any given time. The selected traces correspond to system sizes ranging from 128 nodes in SDSC SP2 to 163840 cores in ANL Intrepid and very low system utilization of 14.9% in DAS2 to high utilization of 85.2% in CTC SP2. We also used traces of two Top500 systems – Intrepid, a BlueGene/P system with 163840 cores at the ANL (number 67 in Nov'13 list) and the CEA Curie supercomputer with 77184 cores (number 20 in Nov'13 list). In Table IV, we show the scheduling algorithms used by these systems. We obtained this information through private communications with system administrators and archival information published by the supercomputing sites. Because our framework does not assume any knowledge of the scheduling algorithm and relies entirely on the history of job submissions for predictions, we are able to obtain useful predictions across a wide range of scheduler deployments and different system and usage profiles. Hence, we claim that our results are representative of actual production supercomputer workloads.

For trace-based experiments, we pessimistically discard the first 10,000 jobs in each trace to disregard any site-specific start-up effects. Studies indicate that users warm up the system environment in the early part of the workload and that this workload is not representative of production usage [30].

In this phase, it is also expected that system administrators experiment with and tune parameters to prepare the site ecosystem for capability computing.

Our predictor uses a number of parameters, for example history size, to tune the prediction strategy to the site and workload specific characteristics. The *validation set* used for parameter tuning consists of 4000 jobs starting at job number 16000 in each trace. For obtaining predictions for the validation set, 6000 jobs starting from job number 10000 are considered as history submissions. We identified all the parameters used by our framework, associated each parameter with a set of possible values, and varied each parameter independent of the others to find an optimal configuration of parameters on the validation set. We evaluate predictions from each parameter configuration using average absolute error (AAE) calculated as the average of the absolute differences between actual and predicted waiting times. For a set of possible values for a parameter, the value that gives the minimum AAE is selected as the optimal value. We found that this optimal configuration of parameters varied for different traces. Table V shows the parameters and the range of parameter values we experimented in our sensitivity studies. Table VI shows the AAE obtained for predictions in the validation set in each trace using feature summaries and distributions (Columns 2 and 3). In each trace, the selected statistic (feature summary or distribution) is highlighted in bold.

Additionally, the distributions used in distance computation for each job were analyzed for closeness to the uniform distribution. For measuring the closeness to the uniform distribution, the distribution is normalized by the total number of jobs and χ^2 distance to the appropriately normalized uniform distribution is computed. We average this distance over all distributions for all jobs in the validation set to assign a *non-uniformity* score to a trace. From column 4 in Table VI, we can see that ANL Intrepid, DAS2, and SDSC Paragon have larger deviations from uniformity compared with the other traces. This confirms the observations from the validation set where we find that distributions outperform feature summaries for the same three workloads (columns 2 and 3).

The optimal parameter configuration is used to obtain predictions on the *test set*, which consists of job numbers 20001 to 60000 in each trace. We compare the performance of our approach with two previously proposed predictors, QBETS [6] and IBL [19], which were described in Section 2. The predictions are evaluated using different metrics that highlight various aspects of the prediction quality.

Table V. Parameters and ranges of values.

Parameter name	Range of values
Size of history set	[2000–6000]
Number of bins used in the distribution	[5–50]
Density-based clustering – $k\%$, f, ϵ, *minPts*	[1–5], [0.10–0.90],[0.05–0.1], [2–5]
SDM – window size, distance threshold	[0.01–0.1], [0.3–0.6]
Ridge regression – maximum distance of history jobs	[0.4–1.0]
Weighted average – number of neighbors	[1–20]

Table VI. Average absolute error (seconds) for different statistics on the validation set.

Trace name	Feature summary	Distribution	Non-uniformity
CTC SP2	**22648.11**	31003.46	0.54
SDSC SP2	**12013.29**	13069.94	0.62
HPC2N	**19275.88**	20594.39	0.55
SDSC Blue	**12063.72**	16038.08	0.67
CEA Curie	**3575.11**	4080.33	0.69
ANL Intrepid	13530.11	**11413.95**	**0.74**
DAS2	41.45	**31.23**	**0.79**
SDSC Paragon	3685.58	**2688.75**	**0.82**

The fourth column is the non-uniformity score for each workload.
For each trace, the selected statistic type is marked in bold.

4.2. Results

We evaluate our predictions on the test set using AAE. The AAE of a job is independent of the response time of the job. However, from a user's perspective, an error of 20 min may be more acceptable for a job with response time of 10 h than a job with response time of 100 min, the latter case representing a more serious prediction error. To include this bias in the error computation, we also compute the percentage difference in predicted and actual response times for each job, where response time is the sum of queue waiting time and execution time. For the execution time, we consider the predicted execution time to be equal to the actual execution time. Hence the percentage predicted error in response time is calculated as $\frac{|predicted\_wait\_time - actual\_wait\_time|}{actual\_response\_time}$ [22].

Table VII shows the average percentage prediction error in response time, PPE_{rt}, and AAE values for different supercomputing job traces for three methods, namely, QBETS, IBL, and the proposed method Qespera. We find that the AAE of our Qespera method is up to 22% and 95% smaller than IBL and QBETS, respectively. We can also observe that the AAE of Qespera is at least 1 h less than IBL for SDSC SP2, HPC2N and ANL Intrepid. Analyzing the cumulative distribution of the absolute errors, we found that across traces, 57–98% of the jobs have an absolute error less than 1 h.

The average percentage prediction error of our Qespera method is up to 375 times smaller than QBETS. In all except two cases, it is between 41% and 4.36 times smaller than IBL. In SDSC-Blue, the average PPE_{rt} is the same in both the IBL and our method. In CEA Curie, we find that while the AAE of our method is 14% smaller than with IBL, the average PPE_{rt} is 12% higher.

We found that the value of average percentage prediction error in response time is dominated by large PPE_{rt} values for jobs with small actual response times. For CTC, Figure 6 shows that jobs with small response times have PPE values (for Qespera) as high as 500. To reduce this artificial bias in the metric, we adopt the approach used in the definition of the bounded slowdown metric [31] to control the range of PPE_{rt}. We lower bound the actual response time in the denominator of the PPE_{rt} equation by a constant value to define the bounded percentage prediction error in response time as $PPE_{rt}^{(\tau)} = \frac{|predicted\_wait\_time - actual\_wait\_time|}{\max\{actual\_response\_time, \tau\}}$. Because the behavior of the metric depends on τ, we investigate the values of the metric for $\tau = 600$ and 1200 s.

In Tables VII and VIII, we can see that $PPE_{rt}^{(600)}$ and $PPE_{rt}^{(1200)}$ reduce the percentage prediction error by up to 22 and 31 times, respectively, for our Qespera method. For example, for SDSC-Paragon, while the PPE_{rt} shown in Table VII is 2.87, the $PPE_{rt}^{(1200)}$ shown in Table VIII is only 0.19. This large difference in the average percentages indicates the degree of skewness induced by large PPE_{rt} for small response time jobs. We once again find that the bounded PPE_{rt} values of our Qespera method are generally lower than with QBETS and IBL, except in SDSC Blue where $PPE_{rt}^{(600)}$ of our method is only 3% higher than with IBL. Overall, we see that $PPE_{rt}^{(1200)}$ of Qespera is up to 2.5 times lesser than IBL and up to 93 times lesser than QBETS. The varying performance differences of our method over IBL and QBETS arise from the different workload

Table VII. Prediction accuracy.

	QBETS		IBL		Qespera	
Log name	Avg. PPE_{rt}	AAE (hours)	Avg. PPE_{rt}	AAE (hours)	Avg. PPE_{rt}	AAE (hours)
SDSC SP2	391.61	44.37	51.48	9.42	11.81	7.84
CTC SP2	148.28	22.98	8.67	4.69	2.69	4.05
HPC2N	3020.44	30.99	40.82	7.65	23.29	5.97
ANL Intrepid	35.15	25.87	0.93	5.68	0.55	4.52
CEA Curie	2444.03	19.19	18.21	3.10	20.35	2.65
SDSC Blue	319.93	29.44	3.56	5.30	3.56	4.83
DAS2	32.03	0.13	3.13	0.04	1.07	0.03
SDSC Paragon	1078.39	14.99	6.63	0.88	2.87	0.69

QBETS, Queue Bounds Estimation from Time Series; IBL, instance-based learning; AAE, average absolute error.

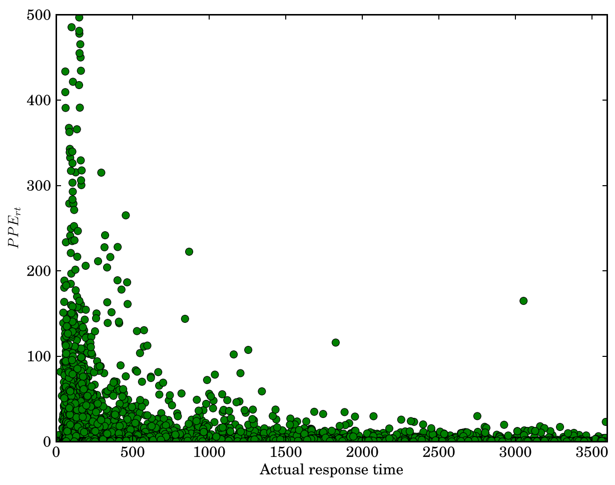


Figure 6. PPE_{rt} in the Cornell Theory Center.

Table VIII. Bounded percentage prediction error of response time.

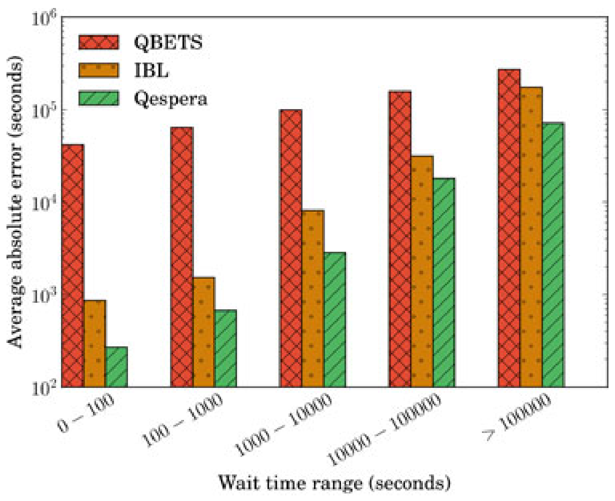
	QBETS		IBL		Qespera	
Log name	Avg. $PPE_{rt}^{(600)}$	Avg. $PPE_{rt}^{(1200)}$	Avg. $PPE_{rt}^{(600)}$	Avg. $PPE_{rt}^{(1200)}$	Avg. $PPE_{rt}^{(600)}$	Avg. $PPE_{rt}^{(1200)}$
SDSC SP2	87.55	50.96	8.55	4.96	3.44	2.10
CTC SP2	45.62	27.30	2.75	1.77	1.13	0.81
HPC2N	52.92	31.83	1.72	1.21	1.05	0.75
ANL Intrepid	27.69	11.37	0.80	0.56	0.46	0.31
CEA Curie	65.93	36.49	1.01	0.64	0.99	0.60
SDSC Blue	69.07	40.55	1.47	1.10	1.52	1.08
DAS2	0.51	0.27	0.08	0.05	0.03	0.02
SDSC Paragon	31.91	17.70	0.65	0.38	0.3	0.19

QBETS, queue bounds estimation from time series; IBL, instance-based learning; AAE, average absolute error.

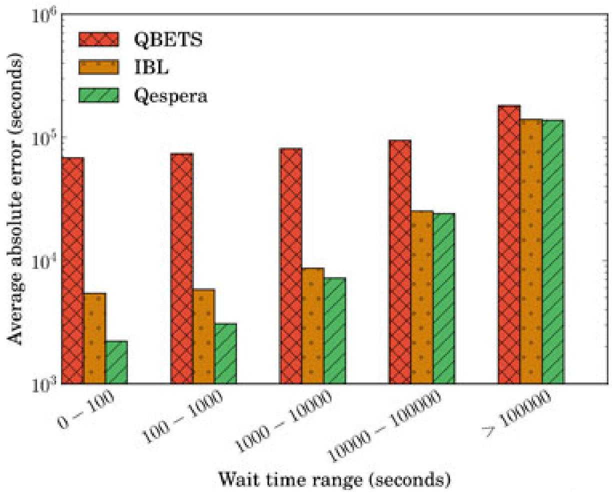
characteristics and scheduling algorithms of the systems. In general, the adaptive nature of our algorithm will allow us to exhibit superior performance compared with IBL in workloads with frequent variations in workload characteristics and scheduling policies.

In addition to showing aggregate results over all the jobs in the test set, we study the errors incurred in different job classes based on the wait time. We divide the jobs into five classes: [0-100 s], [100–1000 s],. . . , [>100,000 s] and analyze the AAE per class. Figure 7 shows the improvement in AAE obtained over IBL and QBETS for different wait time categories for ANL Intrepid and CTC supercomputing traces. Because the graphs show error in log scale, small differences among the methods as seen in the graphs are significant. For example, in the [>100,000 s] class in Figure 7(a), the AAE of our Qespera method is about four times lesser than QBETS and two times lesser than IBL.

To further analyze the improvements, we use a heat map, which is a graphical representation of a two-dimensional histogram with a color mapping for bin frequency. In Figure 8(a) and (b), the heat maps for predictions by IBL and our Qespera method are shown using a log color scale where the darkest bin has 50 times more jobs than the lightest bin. In these plots, good predictions are close to the diagonal where the true and predicted wait times are the same. We see that the predictions for jobs with waiting time in the range [10^3–$10^5 s$] are closer to the diagonal in case of our Qespera method than for IBL. IBL has a significant number of underpredictions and more outliers compared



(a) ANL Intrepid



(b) CTC SP2

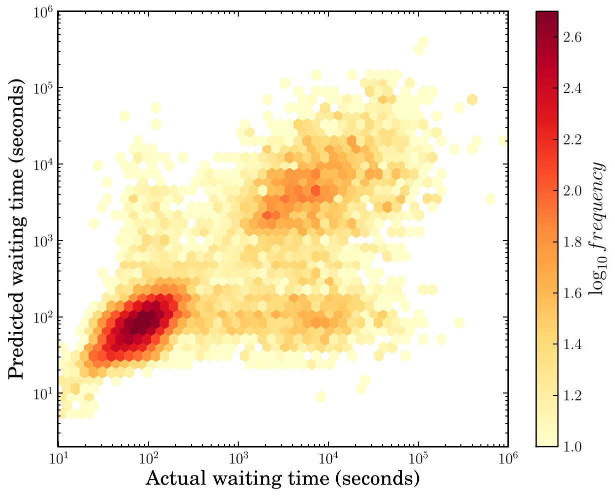
Figure 7. Prediction errors in different wait time categories. (a) ANL Intrepid (b) CTC SP2.

with our Qespera method. Thus, our method gives not only improved prediction accuracy for the overall aggregated results but also improvements for different classes of jobs.

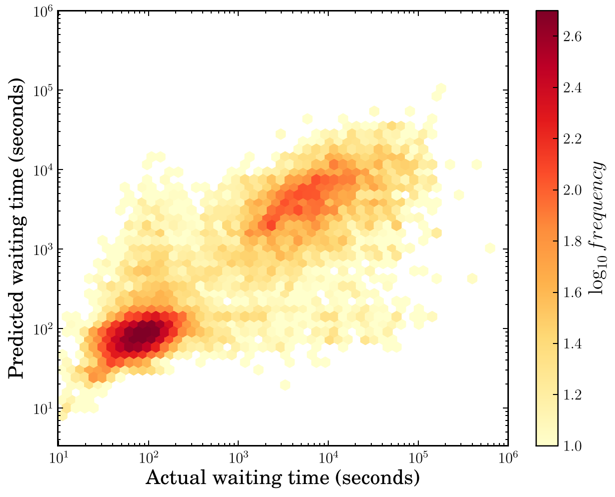
4.3. *Adaptivity*

To illustrate adaptive prediction model selection, we determined the number of predictions made in each trace by each prediction model on the test set. In Table IX, we see that a majority of the jobs use SDM, followed by ridge regression and WA.

As explained earlier, we adaptively select either feature summaries or distributions for a trace based on the relative performance of these two statistics on the validation or training set for the trace. We verified if the inferences made based on the validation set also hold for the test set for two traces, namely, ANL Intrepid and CTC traces. As shown in Table VI, using distributions gave smaller errors for ANL, and feature summaries gave smaller errors for CTC trace on the validation sets. We obtained predictions on the test sets for both the traces using both the statistics, namely, distributions and feature summaries. Similar to the observations in the validation sets, we found that in the test sets, distributions gave 47% reduction in ${PPE_{rt}}^{(1200)}$ and 5% reduction in AAE over feature summaries for ANL Intrepid and that feature summaries gave 54% reduction in ${PPE_{rt}}^{(1200)}$ and 20% reduction in AAE over distributions for CTC trace.



(a) IBL



(b) Qespera

Figure 8. Heat maps for predictions in ANL Intrepid. (a) IBL. (b) Qespera.

4.4. Analyzing large prediction errors: Argonne National Laboratory case study

We tried to analyze the reasons for large prediction errors, that is, *bad predictions*, obtained for some jobs. We used the ANL Intrepid system as a case study, because in addition to the job workload traces, failure logs were also maintained for this system. Moreover, we were able to clarify the possible reasons for some anomalies we noticed in the workload traces through communications with the system administrators of this system.

We define a *goodness* metric for our predictions in this workload. To define *good predictions*, we divide the queue waiting times into different clusters or bins. To obtain these bins, we select all jobs in the training set and identify clusters of wait times using k-means clustering. The *gap statistic*, which compares the intra-cluster dispersion of points to a null reference [32], was used to compute the optimal number of clusters in the training set. The cluster boundaries were rounded off to the nearest hour to obtain a set of continuous intervals. Figure 9 shows the cluster or bins obtained for the ANL Intrepid system.

To define the goodness metric, we first find the bin in which the predicted wait time falls. We then define a *prediction range* centered around the predicted wait time with the width of the range equal to the bin width. A prediction is termed *good* if the actual wait time falls within this prediction range. For example, we found that the clustering method identified a cluster of jobs with wait times

Table IX. Adaptive predictor selection of Qespera.

Trace name	SDM (%)	Ridge regression (%)	*k*-weighted average (%)
SDSC SP2	87	13	0
CTC SP2	81	18	1
HPC2N	89	10	1
CEA Curie	95	4	1
ANL Intrepid	93	6	1
SDSC Blue	85	13	2
DAS2	100	0	0
SDSC Paragon	100	0	0

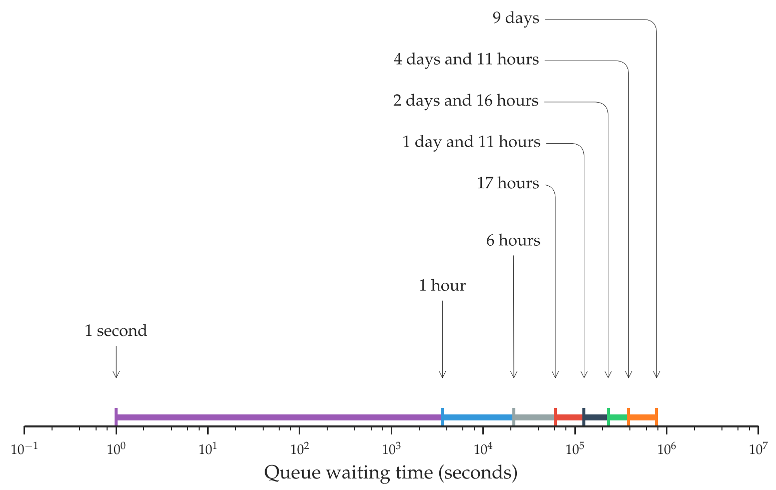


Figure 9. Clusters of queue waiting times observed in ANL Intrepid. The duration spanned by a cluster is marked by a line segment of a particular color.

between 1 and 6 h. For some job in Intrepid, if the predicted wait time is 4 h, then the prediction range is 1.5–6.5 h. We consider the prediction for the job as *good* if the actual waiting time is within this range.

Using the aforementioned metric, 70% of the predictions made by Qespera were found to be good predictions for the ANL Intrepid system. For these good predictions, Qespera obtained an AAE of only 15 min and percentage prediction error of 14%. For the remaining 30% of the jobs, we examine a series of system-specific conditions as possible reasons for the bad predictions.

Condition 1: Intrepid uses a priority scheduler that computes priority of the jobs using the request size, ERT, and already queued time. Tang *et al.* [28] show that the priority computations used for Intrepid make the scheduler similar to a shortest-job-first algorithm. Hence, the wait time of a job is influenced by future jobs that request lesser wall clock time.

Condition 2: System failures result in certain partitions becoming unavailable for scheduling. Intrepid is a BlueGene/P system that uses a partitioned torus network for communication. The node partitioning scheme allows only partitions of uniform length in each dimension and jobs can only use a set of adjacent racks. During interactions with the system administrators, we learned that even single node failures in such a partitioning scheme can lead to long delays in resource allocation for queued jobs even though the resources are available.

Condition 3: Intrepid also follows a scheduled maintenance period on every other Monday between 9:00 AM and 5:00 PM Central Time, during which jobs are not run on the system. Queued jobs during the maintenance period can suffer large delays, which are difficult to predict. Because the queue is not expected to be in steady state just after a maintenance period is over, jobs arriving just after the scheduled period can also suffer from bad predictions.

Condition 4: On every Monday, all jobs waiting in the *prod − capability* queue are promoted to the highest priority queue. This artificial tweaking of job priority is hard to factor in the prediction algorithm and can lead to wrong predictions.

Condition 5: Jobs in the *prod − long* queue are constrained to run on a fixed set of 16 racks, unlike other queues where jobs can run on any of the available 40 racks (including the 16 racks of *prod − long*). These jobs tend to have small request size of (512–4096 nodes) and large ERTs (>6–12 h). When the scheduler cannot run the job at the head of the production queue, it drains a partition that can accommodate the job at the earliest. In this context, the large ERT of *prod − long* jobs makes it harder for the scheduler to consider them for backfilling on draining partitions. The partitioning constraints of the BlueGene architecture aggravate this problem if the draining partition corresponds to the racks allocated to *prod − long*.

To study the effect of these conditions on the bad predictions, we count the number of jobs corresponding to each condition. Condition 1 can be easily checked by counting the number of future shorter job arrivals for each target job. To analyze the effect of system failures on predictions for condition 2, we use the reliability, availability, and serviceability log available from the USENIX Computer Failure Data Repository [33] and extract the set of fatal hardware events according to the criteria specified in [34]. If a failure occurs during the waiting period of a job or up to 10 min before the arrival of a job, we claim that the wait time of the job is affected by the failure. We choose 10 min because the maximum boot time for a partition in ANL Intrepid is 11 min. So a failure that requires only a node restart can take up to 11 min. To study conditions 3 and 4, we check whether the job is queued on a maintenance day or if its waiting period includes a maintenance period. For condition 5, we simply check whether the job belongs to *prod − long* queue.

In Table X, we show the percentage of good/bad predictions affected by each condition. For each job, we also counted the number of shorter jobs that arrived during the waiting period of the job and started executing before the job. The last row of the table shows that for jobs with good predictions, the number of future short job arrivals is almost 10 times less than the jobs with bad predictions. We

Table X. Analysis of prediction errors in ANL Intrepid.

Reason	Good predictions	Bad predictions
Monday cases	7.1%	17.2%
Jobs in prod-long queue	3.6%	21.0%
Failure cases	3.0%	29.1%
Jobs affected by future arrivals	24.4%	88.2%
Average number of future small jobs	15	149

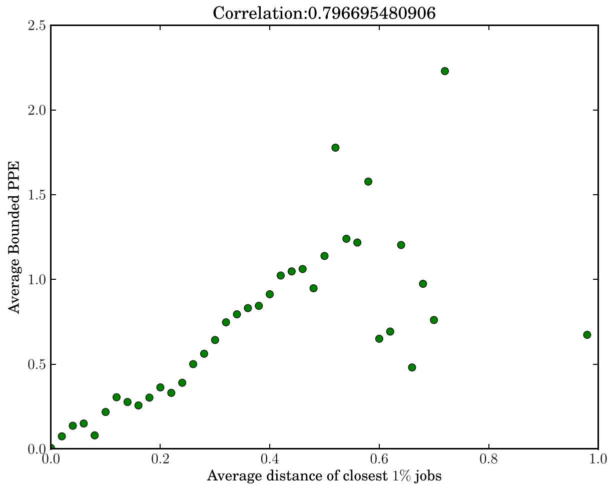


Figure 10. Relationship between prediction error and distance of the nearest neighbors of jobs in ANL Intrepid.

also find that the other conditions also contributed significantly to the bad predictions. Using all the conditions, we were able to account for 91% of the bad predictions.

For Intrepid, we also studied the relationship between the distance of the NN of a job and the prediction error. Recall that the distance of the NN is used to decide the prediction model used by our algorithm. To study the relationship, we divided the distance range $[0, 1]$ into multiple bins and averaged the error observed and the distance of the closest 1% of the neighbors for jobs in each bin. In Figure 10, we see that the average error depends linearly on the average distance of the NN till a distance value of 0.5. The bins that have distances of more than 0.5 have very few jobs, and these points exhibit high variability in error. The Pearson correlation of the error versus the average distance is 0.79. This signifies a strong linear relationship and indicates that whenever the history contained enough similar jobs for a target job, our predictor obtained small error. The high errors for jobs that have far NN indicates that an insufficient history of similar jobs is the primary cause of prediction error.

5. DEPLOYMENT IN OUR DEPARTMENTAL FACILITY

We have implemented our Qespera framework on Tyrone, an 800-core cluster in our department. This cluster consists of 17 rack mounted nodes with 32 or 64 cores each, interconnected using Infiniband. Tyrone has six queues as shown in Table XI. The batch queue is an intermediate queue that accepts job submissions from the users and routes them to the appropriate scheduling queue based on their parameters. Job submissions to these queues are handled by the Torque Resource Manager, which is based on OpenPBS [35]. The scheduling policy is FCFS with limits on the number of queued and running jobs per user.

To deploy our framework, we implemented a history collection routine that parses the log files generated by Torque to gather information about previous job submissions. Torque produces two kinds of logs that are of interest to us: server_logs and sched_logs. Torque generates one server and sched log per day. For each previous job submission, the server_log corresponding to the day when the job finished execution contains the queued time, submitting user's name, executable name, queue name, and finish time of the job. For each previous job submission, the sched_log corresponding to the day when the job started running has the start time of the job. By co-analyzing these daily logs, we can reconstruct an accurate history of job submission, start, and termination events on the system. For each node, Torque maintains a status field that sets to *free* if the node is available for scheduling, *job-exclusive* if all cores in the node are processing jobs, and *down* if the node does not respond to Torque for a pre-determined time. When a node is marked as down by Torque, a corresponding message is logged in the day's server_log.

Our framework predicts the queue waiting time for each job that is currently queued. To obtain the list of currently queued jobs, we use the PBS command *qstat -f*. This command provides the necessary attributes like job request size, ERT provided by the user, user name, and queue name. For each job, this information is used as input to our prediction algorithm to obtain the predicted queue waiting time. From our experience with the workload of ANL Intrepid, we know that predictions delivered during machine failure events can be error-prone. Hence, we use the node status maintained by Torque to turn off our predictor when one or more nodes have failed. While parsing a server log, if our log collection routine detects that a node is currently down, it turns off the

Table XI. Batch queues in Tyrone cluster.

Queue	CPU cores	Maximum wall time	Maximum running jobs per user	Maximum queued jobs per user
batch	—	—	—	11
idqueue	0 to 32	2 h	1	1
qp32	32	1 day	2	1
qp64	64	1 day	1	1
qp128	128	1 day	1	1
qp256	256	1 day	2	5

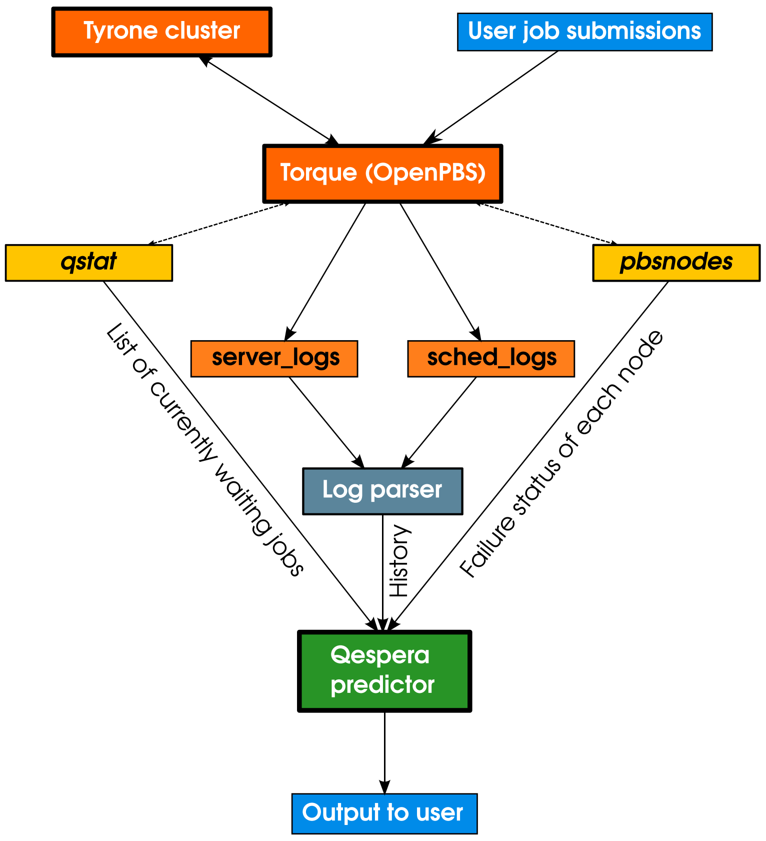


Figure 11. Prediction of queue waiting times in our department cluster.

queue waiting time predictor. Using the command *pbsnodes -a*, it periodically monitors the state of the failed node to turn on the predictor when the node's state is changed from down to free or job-exclusive. The overall framework for prediction of queue waiting times on Tyrone is shown in Figure 11.

Using this prediction mechanism, we have obtained queue waiting time predictions for a set of 23,000 jobs during a period of 2 years from December 1, 2012 to Dec 19, 2014. Overall, our predictions have an AAE of 7338.1 s or 2.03 h. We also observed that the average percentage prediction error is 30.1%. The average queue waiting time on this cluster is 13.5 h. In comparison, our prediction error of 2 h is much lesser and allows users to plan their experiments at the moment of job submission. These predictions are available to users through a web interface that lists the queue waiting time of all currently waiting jobs. The interface also allows users to query the waiting time of potential job submissions. Using this mechanism, users can tune job parameters like request size or ERT to yield faster turnaround time.

6. DISCUSSION

With the aim of maximizing the usability of our predictor, we have chosen job and user features that are applicable across different HPC systems and workload characteristics. In particular, all our features require only information available in the SWF. The accuracy of the features used in our prediction model depends upon the accuracy of the data available in the SWF logs used in our experiments. In general, it is true that log data can have various errors including incomplete or inconsistent data fields, errors in the recorded information, spurious job submission patterns, and missing data regarding system downtime. In the PWA, SWF logs are obtained by parsing historical logs maintained by scheduling systems like PBS and LSF. A subsequent clean-up procedure is used to alleviate some of the aforementioned issues [36]. To the best our knowledge, the information available in these cleaned logs is an accurate record of the jobs processed in the systems used in our study. For a specific system, it is possible to improve the prediction accuracy further using system-specific information. For example, in systems where machine reservations are allowed, we can obtain better predictions by carefully considering the attributes associated with the reservation.

The traces include ERT and the actual run time of the job. While ERTs supplied by the user are generally known to be gross overapproximations, the overall average errors in ERTs do not vary drastically across job submissions over time. When our system makes prediction for a target job under a given queue and processor execution states containing ERTs of jobs with high errors, the predictions are based on history jobs whose states have similar errors for ERTs. At any instant, the job and system state used by our predictor is exactly the state used by the batch scheduler to make a scheduling decision. Hence, when we obtain a prediction for a job submitted at the current system state using a similar historical job submission instant, we assume that the scheduler will make a similar decision at both instants of time. Hence, we do not explicitly consider ERT errors to correct our predictions.

While the concept of clustering history jobs based on similarity and history-based predictions used in the system can be extended to predicting run times, in a separate framework, we are building for predicting run times, we are using different set of features including user and job IDs along with a function that captures the variation of run times with the system loads.

7. CONCLUSIONS AND FUTURE WORK

Prediction of queue waiting times is a beneficial service for users of batch parallel systems. Queue waiting time predictions can be used for planning experiments, choosing job parameters, which can reduce job turnaround times, and for resource selection among multiple supercomputers. Queue waiting time is hard to predict because of its dependence on job parameters, system state, scheduling algorithm and policies, and external factors like system maintenance periods and node failures.

In this paper, we have developed an integrated adaptive framework to predict queue waiting times of jobs submissions for jobs submitted to batch queues in parallel systems. We have proposed a distance metric that allows us to assign a real valued distance to each history job based on its similarity to the target job. Our distance metric is based on distributions or feature summaries, which include job characteristics, current queue and processor characteristics, and the characteristics related to the submitting user. Based on the distance values and a spatial clustering algorithm, we have developed a strategy to pick a suitable prediction model for each incoming job, based on the prevalent job and system characteristics. When we find that a job has very similar history submissions, we use an SDM scheme that computes the prediction as the weighted mean of a close dense cluster of jobs. To improve the quality of prediction, we remove outliers using a density-based clustering algorithm. For jobs that do not have sufficient similar history, we use ridge regression or k-WA to obtain the prediction. It is noteworthy that our prediction methods are independent of the scheduling algorithm used in the supercomputer and do not assume any knowledge of the policies used by the system administrators.

Our experiments conducted using trace-based simulation with workload traces demonstrate the effectiveness of our approach. We exhibit significant improvement over previous efforts, and in one workload trace, we obtain an empirical characterization of the reasons for large prediction errors. Integration of our framework with OpenPBS followed by its deployment in a cluster in our home department also shows that we can obtain useful queue waiting time predictions in production HPC environments.

As part of future work, we plan to integrate our framework with predictions of execution time of a job. This integration will allow us to devise automatic strategies for selecting job parameters that can minimize response time. We are also keen on using our predictions for automatic workflow scheduling in large-scale systems. We also plan to integrate our predictor with many popular scheduling frameworks for easy deployment in other supercomputing sites.

ACKNOWLEDGEMENTS

The authors would like to acknowledge support from the Department of Science and Technology (DST), India SR/S3/EECE/0095/2012.

REFERENCES

1. IBM LoadLeveler. (Available form: http://www-03.ibm.com/systems/software/loadleveler) [Accessed on December 2014].
2. PBS Works. (Available from: http://www.pbsworks.com) [Accessed on December 2014].
3. IBM Platform LSF. (Available from: http://www-03.ibm.com/systems/services/platformcomputing/lsf.html) [Accessed on December 2014].
4. Maui scheduler. (Available from: https://www.adaptivecomputing.com) [Accessed on December 2014].
5. Parallel Workloads Archive. (Available from: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html)[Accessed on December 2014].
6. Nurmi DC, Brevik J, Wolski R. BETS: Queue bounds estimation from time series. In *Proceedings of the 2007 ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07. ACM: New York, NY, USA, 2007; 379–380.
7. TeraGrid. (Available from: https://www.xsede.org/tg-archives)[Accessed on December 2014].
8. Karnak Prediction Service. [Online, December 2014].
9. XSEDE. (Available from: https://www.xsede.org/tg-archives) [Accessed on December 2014].
10. Nurmi D, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06. ACM: New York, NY, USA, 2006; 29–29. DOI: 10.1145/1188455.1188579.
11. Smith W, Foster I, Taylor V. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing* September 2004; **64**(9):1007–1016.
12. Smith W, Taylor VE, Foster IT. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS/SPDP '99/JSSPP '99. Springer-Verlag: London, UK, 1999; 202–219.
13. Feitelson DG, Rudolph L, Schwiegelshohn U. Parallel job scheduling – a status report. In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'04. Springer-Verlag: Berlin, Heidelberg, 2005; 1–16.
14. Downey AB. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Symposium on Parallel Processing*, IPPS '97. IEEE Computer Society: Washington, DC, USA, 1997; 209–218.
15. Cobalt resource manager. (Available from: https://trac.mcs.anl.gov/projects/cobalt) [Accessed on 6 November 2015].
16. Tang W, Desai N, Buettner D, Lan Z. Analyzing and adjusting user runtime estimates to improve job scheduling on the blue Gene/P. In *International Symposium on Parallel and Distributed Processing*, IPDPS. IEEE, 2010; 1–11.
17. Tang W, Desai N, Buettner D, Lan Z. Job scheduling with adjusted runtime estimates on production supercomputers. *Journal of Parallel and Distribued Computing* 2013; **73**(7):926–938.
18. Smith W. Prediction services for distributed computing. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE international*. IEEE: Long Beach, USA, 2007; 1–10.
19. Li Hui, Chen J, Tao Y, Gro D, Wolters L. Improving a local learning technique for queue wait time predictions. *IEEE International Symposium on Cluster Computing and the Grid* 2006; **0**:335–342.
20. Li H, Groep D, Wolters L. Efficient response time predictions by exploiting application and resource state similarities. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05. IEEE Computer Society: Washington, DC, USA, 2005; 234–241.
21. Li H, Groep D, Wolters L. Mining performance data for metascheduling decision support in the grid. *Future Generation Computer Systems* 2007; **23**(1):92–99.
22. Kumar R, Vadhiyar S. Identifying quick starters: towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs. In *Job Scheduling Strategies for Parallel Processing*, Cirne W, Desai N, Frachtenberg E, Schwiegelshohn U (eds). Springer-Verlag, 2012; 196–215. Lect. Notes Comput. Sci. vol. 7698.
23. CTC-96-HT08: Changes to EASY-LL Job queuing algorithm and policy. (Available from: http://web.archive.org/web/19970614012203/http://www.tc.cornell.edu/UserDoc/HotTips/1996/ht08.html) [Accessed on December 2014].
24. Standard workload format [SWF]. (Available from: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html) [Accessed on December 2014].
25. Cornell Theory Center User Documentation: Modifying LoadLeveler Jobs to Run with EASY-LL. (Available from: http://web.archive.org/web/19970614002518/http://www.tc.cornell.edu/UserDoc/SP/Batch/Easy/)[Accessed on December 2014].
26. The Batch System on Seth at HPC2N. (Available from: web.archive.org/web/20031207233609/http://www.hpc2n.umu.se/support/userguide/Seth/batchsystem.html) [Accessed on December 2014].
27. *The DAS2 5-Cluster Grid Logs*. (Available from: http://www.cs.huji.ac.il/labs/parallel/workload/l_das2/index.html) [Accessed on December 2014].
28. Tang W, Lan Z, Desai N, Buettner D. Fault-aware, utility-based job scheduling on blue Gene/P systems. In *Cluster*. IEEE: New Orleans, USA, 2009; 1–10.

29. SLURM at CEA. (Available from: slurm.schedmd.com/SUG13/cea-site-report-0.6.pdf) [Accessed on December 2014].
30. Joubert W, Su S-Q. An analysis of computational workloads for the ORNL Jaguar system. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12. ACM: New York, NY, USA, 2012; 247–256.
31. Feitelson DG. Metrics for parallel job scheduling and their convergence. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01. Springer-Verlag: London, UK, 2001; 188–206.
32. Tibshirani R, Walther G, Hastie T. Estimating the number of clusters in a data set via the Gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 2001; **63**(2):411–423.
33. USENIX. *CFDR: The Computer Failure Data Repository*. (Available from: http://www.usenix.org/cfdr) [Accessed on December 2014].
34. Zheng Z, Yu L, Tang W, Lan Z, Gupta R, Desai N, Coghlan S, Buettner D. Co-analysis of RAS Log and job log on blue Gene/P. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE: Anchorage, USA, 2011; 840–851.
35. TORQUE resource manager. (Available from: http://www.adaptivecomputing.com/products/open-source/torque/) [Accessed on December 2014].
36. Feitelson DG, Tsafrir D, Krakov D. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing* 2014; **74**(10):2967–2982.