Identifying Quick Starters: Towards an Integrated Framework for Efficient Predictions of Queue Waiting Times of Batch Parallel Jobs

Rajath Kumar and Sathish Vadhiyar

Supercomputer Education and Research Center, Indian Institute of Science, Bangalore, India
{rajath@ssl.serc.iisc.in,vss@serc.iisc.in}

Abstract. Production parallel systems are space-shared and hence employ batch queues in which the jobs submitted to the systems are made to wait before execution. Thus, jobs submitted to parallel batch systems incur queue waiting times in addition to the execution times. Prediction of these queue waiting times is important to provide overall estimates to the users and can also help metaschedulers make scheduling decisions. Analyses of the job traces of supercomputers reveal that about 56 to 99% of the jobs incur queue waiting times of less than an hour. Hence, identifying these quick starters or jobs with short queue waiting times is essential for overall improvement on queue waiting time predictions. Existing strategies provide high overestimates of upper bounds of queue waiting times rendering the bounds less useful for jobs with short queue waiting times. In this work, we have developed an integrated framework that uses the job characteristics, and states of the queue and processor occupancy to identify and predict quick starters, and use the existing strategies to predict jobs with long queue waiting times. Our experiments with different production supercomputer job traces show that our prediction strategies can lead to correct identification of up to 20 times more quick starters and provide tighter bounds for these jobs, and thus result in up to 64% higher overall prediction accuracy than existing methods.

Keywords: Queue Wait Times, High Performance Computing, Batch Systems, Prediction, Scheduling

1 Introduction

Production parallel systems in many supercomputing sites are batch systems that provide space sharing of available processors among multiple parallel applications or jobs. Well known parallel job scheduling frameworks including IBM Loadleveler [1], PBS [2], Platform LSF [3] and Maui scheduler [4] are used in production supercomputers for management of jobs in the batch systems. These frameworks employ batch queues in which the jobs submitted to the batch systems are queued before allocation by a batch scheduler to a set of available processors for execution. Thus, in addition to the time taken for execution, a job submitted to a batch queue incurs time due to waiting in the queue before allocation to a set of processors for execution.

Predicting queue waiting times of the jobs on the batch systems will be highly beneficial for users. The predictions can be used by a user for various purposes including planning management of his jobs and meeting deadlines, considering migrating to other queues, systems or sites at his disposal for application execution when informed of possible high queue waiting times on a queue, and investigating alternate job parameters including different requested number of processors and estimated execution times. Such predictions can also be efficiently used by a metascheduler to make automatic scheduling decisions for selecting the appropriate number of processors and queues for job execution to optimize certain cost metrics, and help reduce the complexities associated with job submissions for the users. The decisions by the user and metascheduler using the predictions can in turn result in overall load balancing of jobs across multiple queues and systems. Such predictions are also highly sought after in the production batch systems. For example, predictions of queue waiting times are available in production systems of TeraGrid [5]. These show the importance of accurate queue wait time prediction mechanisms for the users submitting their jobs to batch systems.

Analyses of widely used job traces in supercomputer sites reveal the presence of large number of jobs that incur short queue waiting times. Table 1 shows statistics for eight different supercomputer job traces we use in this work. All the eight traces were cleaned versions obtained from Feitelson's workload archive [6]. The last column of the table shows the percentages of the number of jobs with queue waiting times of less than or equal to 1 hour. We find that most of the jobs, particularly 56-99% of the total number of jobs, submitted to a system incur queue waiting times of less than or equal to 1 hour. We refer to these jobs as *quick starters*. Correct identification and good predictions of these quick starters that form a majority are hence essential for overall accuracy of the prediction system.

Table 1. Supercomputing Log Details

Trace	**Duration (in months)**	**Total no of jobs**	**% quick starters**
CTC SP2	11	77222	56
ANL Intrepid	8	68936	65
LANL CM5	24	122060	94
HPC2N	42	202876	57
SDSC Paragon 95	12	53970	88
SDSC Blue	32	243314	70
SDSC SP2	24	59725	66
DAS2 fs0	12	225710	99

It is important to note that these quick starters do not necessarily correspond to testing/debugging jobs that are associated with short execution times, and whose predictions are relatively less important. Many systems have separate debug queues for testing/debugging jobs. Our experiments were conducted on general/production queues in which production runs are performed, and where predictions of queue waiting times of the production jobs are required. A significant number of quick starters in these production queues have high execution times. For example, in CTC and ANL production queues, about 30% of the quick starters have runtimes greater than 1 hour and some of them have runtimes as high as 120 hours. Prediction of queue waiting times is challenging due to various factors including diverse scheduling algorithms followed by the job scheduling frameworks, time-varying policies applied for a single queue, and priorities for the jobs. High values of predictions will have more severe impact on the predictions for quick starters than for jobs with long wait times. High overestimations for quick starters can have detrimental effect even on the job submissions to the system. For example, an upper bound of 8 hours for a job that executes for 15 minutes and whose actual waiting time will be 30 minutes can discourage the user from submitting the job to the system that the user would have found suitable for his job in the absence of such overestimation. Hence it is essential to give tight upper bounds especially for these quick starters. In our work, we fix this upper bound as 1 hour for all the quick starters. The assumption is that even if a quick starter's actual waiting time is 5 minutes, this upper bound of 1 hour will not severely discourage the user since the user typically expects waiting times of at least few minutes to an hour in a multi-user system.

The objective for predictions of quick starters is two fold:
• Maximizing *true positives*, i.e., increasing the number of correct identifications of quick starters
• Minimizing *false positives*, i.e., decreasing the number of incorrect identification as quick starters of jobs with long queue waiting times.
The former objective is important for improving the overall accuracy of predictions, while the latter is essential to avoid "misguiding" (or colloquially, "cheating") the user into using the system with the promise of short queue waiting times.

In this work, we have developed an integrated framework, **PQStar** (**P**redicting **Q**uick **Star**ters), for identification and prediction of quick starters. An important aspect of our prediction strategy for quick starters is that it considers the processor occupancy state and the queue state at the time of the job submission in addition to the job characteristics including the requested number of processors and the estimated runtime. The processor and queue states include the current number of free nodes, number of jobs with large request sizes currently executing in the system, and relative difference between the current job and other jobs in the queue in terms of request size and estimated run time. These states are obtained by using a simulator that updates the states during job arrivals and departures. For jobs identified as those with potential long queue waiting times, our framework uses existing strategies [7, 8] for predictions. Our experiments with different production supercomputer job traces given in Table 1 show that our prediction strategies can lead to correct identification of up to 20 times more quick starters and provide tighter bounds for these jobs, and thus result in up to 64% higher overall prediction accuracy than existing methods. Our model was designed not to use dynamic and variable parameters including scheduling algorithms and job priorities. In many cases, it is not practical to obtain/infer job priorities and scheduling algorithms. Scheduling algorithms on batch systems are usually not published, and are not easy to model. Our model primarily uses the job traces, and the job submission states (queue and processor occupancy states). This way, our system can be generic and can be applied to different batch systems with different scheduling and priority policies.

Section 2 presents existing strategies for predictions of queue waiting times. In Section 3, our prediction methodology is described in detail. Section 4 describes the simulation experiments with the supercomputer job traces and presents results related to accuracy in identifying quick starters and overall predictions. This section also compares the performance of our predictions with the existing methods. Section 5 presents a summary of our work and plans for future work.

2 Related Work

There have been two primary efforts in prediction of queue waiting times. They can be broadly classified into *Non-Statistical* and *Statistical* methods.
Non-statistical methods try to simulate the exact scheduling algorithm and decisions which would be made by the scheduler in real time. However, in most production systems, the scheduling algorithms are usually not published and are also difficult to model.

In the works by Smith et al. [9] [10], runtime predictions are derived using similar runs in the history, and these estimates are further used to simulate the scheduling algorithms like FCFS, LWF (Least Work First) and Backfilling [11] to obtain the queue wait times predictions. Another work by Li et al. [12] tries to improve the runtime prediction and simulate the batch system for the Maui scheduler [4]. These efforts consider specific scheduling algorithms for predictions while our effort considers only job traces and hence can work with multiple scheduling algorithms. Moreover, their efforts use runtime estimates for the prediction of queue wait times. The runtime estimates reported in these efforts [9] [10] have high prediction errors from 33% to 74% in many cases, and hence using these estimates to predict queue wait times will lead to large errors in wait time predictions.

The statistical method by Downey [13] used the observation that the cumulative distributions of the execution times of the jobs in the workload can be modeled by using a logarithmic function. After the distribution functions are calculated, two different methods are used to predict when a certain number of nodes will become free and thus when the job waiting at the head of the queue can start. This work considers FCFS Scheduling.

Some statistical methods use time series analysis of queue waiting times for jobs in the history to predict waiting times for submitted jobs. QBETS [14–16] is a system that predicts the bounds on the queue wait times with a quantitative confidence level. QBETS uses a quantile-based approach in which a given quantile in the distribution of the queue waiting times of the jobs is used as an upper bound for the target job's queue waiting time. Since the distribution is not known, a confidence level has to be provided. QBETS uses a predictor based on non-parametric inference, an automated change-point detector, machine-learned, model-based clustering of jobs having similar characteristics, and an automatic downtime detector to identify systemic failures that affect job queuing delay. Thus QBETS handles the effects of varying workloads and customized local queuing discipline. However, QBETS gives conservative upper bound predictions, which leads to large prediction errors especially for quick starters. Also it does not consider the state of the system, and consider only the job characteristics, which we show is insufficient for efficient predictions of queue wait times.

The efforts by Li et al. [7, 8] consider the system states for the prediction of queue waiting times. In their method known as Instance Based Learning (IBL), they use weighted sum of Heterogeneous Euclidean-Overlap Distances between different attributes of two jobs to find the similarities between the jobs. They consider both job characteristics and system parameters for job attributes. They then find such similarity values between the target job and all jobs in the history and choose a subset of most similar jobs in the history, and use their queue waiting times in methods like 1-NN (nearest neighbor) or the n-WA (weighted average of n nearest neighbors) to predict the waiting time of the target job. Their work assumes linear relationship between attribute values and queue waiting times. They also assume fixed weights for predictions of all target jobs. Our work explicitly considers quick starters for predictions. We show that our method gives better predictions of quick starters. By providing tighter estimated bounds for *quick starters* that form a majority of the jobs, our work attempts to improve the overall accuracy of predictions.

3 Methodology

The basis of our method, *PQStar*, for identifying a quick starter job is to establish boundaries in the history of prior job submissions, and to use the similar jobs within the boundaries for prediction. Thus, the most relevant history is used for predicting the target quick starter job. Specifically, PQStar splits the history for a target job into near, mid and long term history based on processor occupancy states. A processor occupancy state at a given instance denotes the allocation of the processors to the jobs executing at that instance. It consists of the number of processors used by each executing job.

3.1 Predictions using Near-term History

For finding the near-term history, PQStar traverses the jobs starting from the target job in the reverse chronological order of job submission times as long as the processor occupancy state at the time of submission of the job in the history is similar to the processor occupancy state at the time of submission of the target job. The earliest job in the history having similar processor occupancy state denotes the near-term boundary and the set of jobs between the target job and the boundary forms the near-term history. For identifying if the target job is a quick starter, PQStar finds jobs in the near-term history with similar characteristics in terms of processor request sizes and estimated run time, and which have started executions. It also checks if none of the jobs in the waiting queue which have arrived after the near-term boundary and which have similar characteristics to the target job, have waited for more than an hour in the queue. If these two conditions are met, PQStar identifies the target job as a quick starter and establishes an upper bound of 1 hour for the target job.

For the purpose of predictions using near-term history, two processor occupancy states are considered similar if the number of jobs with large request sizes that are executing in the two states are the same. We use the executing large jobs to define processor occupancy similarity since jobs that can be backfilled in the remaining processor space and thus incur small queue waiting times are candidate quick starters. Thus the basis of identifying quick starters using near-term history is that by looking at jobs with similar characteristics in the near-term history with similar processor occupancy states and checking if those jobs have potentially been backfilled, it can be predicted if the target job can be backfilled and hence marked as a quick starter. Note that by our definition, two processor occupancy states are also considered similar if there are no large jobs in both the states. For our work, we denote a job as a large job if the request size of the job is at least the next power of two greater than or equal to the square root of the total system size . We define job characteristics of two jobs as similar if their processor request sizes are equal and if the difference between their estimated run times are within an hour. As our experiments will show, this method of using near-term history for predictions yields a large percentage of identifications of quick starters.

3.2 Predictions using Mid-term History

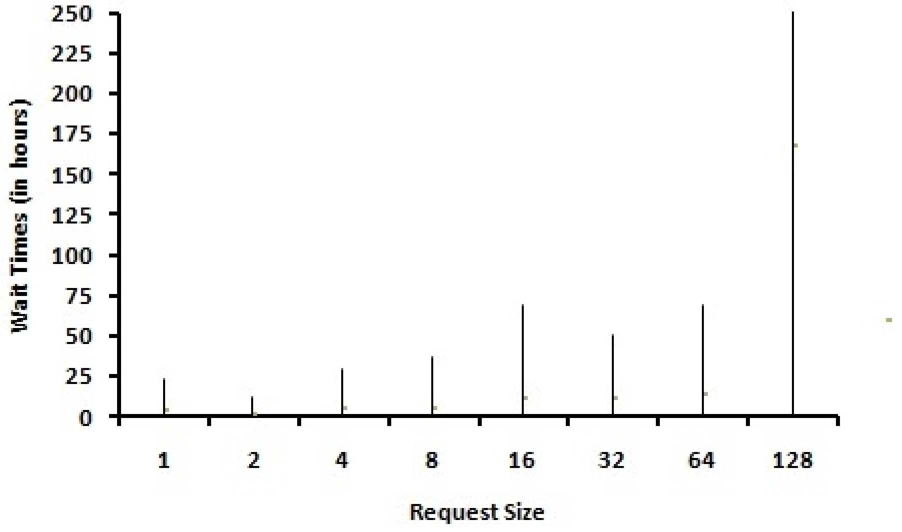
For those jobs for which near-term history cannot be used due to the above mentioned criteria not being met, PQStar traverses the jobs in the reverse chronological order of job submissions from the near-term boundary until the processor occupancy state becomes completely different in terms of executing jobs. In other words, suppose A = {set of jobs in execution at the time of target job's submission} and B = {set of jobs in execution at the time of the history job's submission}, then we draw the mid term boundary at a point where $(A \cap B) = \emptyset$

PQStar marks the *mid-term boundary* after the job when the states become completely different and denotes the set of jobs between the mid and near-term boundaries as mid-term history. For identifying if the target job is a quick starter, PQStar finds jobs in the mid-term history with similar characteristics in terms of processor request sizes and estimated run time, and which have started executions. It also checks if none of the jobs in the waiting queue which have arrived after the mid-term boundary and which have similar characteristics to the target job, have waited for more than an hour in the queue. However, unlike for predictions with near-term history, these conditions alone are not sufficient for predictions with mid-term history since the processor states in mid-term are less similar to target job than those in near-term. We found that jobs satisfying the conditions had widely varying queue waiting times. Hence we introduce three extra criteria for predictions with mid-term history: *request size criterion*, *estimated run time (ERT) criterion*, and *free nodes criterion*.

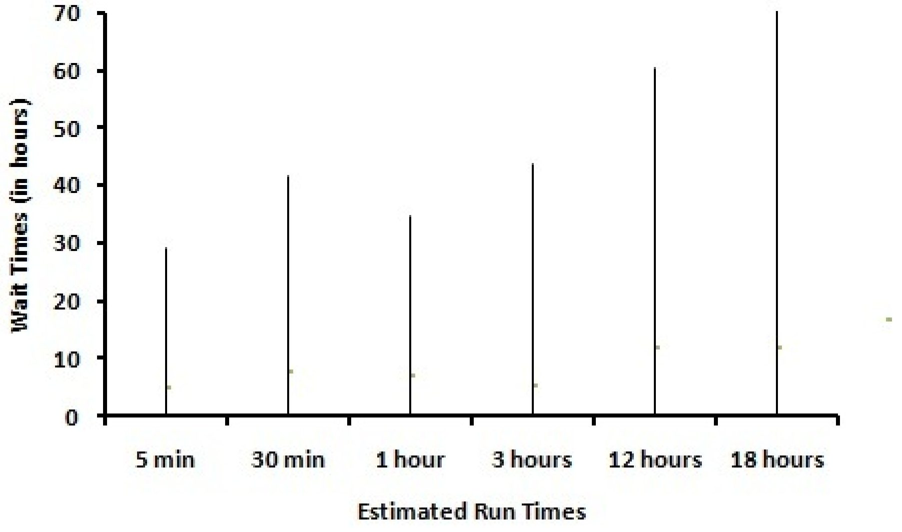
Request Size Criterion: One useful criterion we found is to rank the target job in terms of its request size among the jobs in the queue at the time of the job submission. Specifically, we calculate the metric $jobrank_{reqsize}$ using the position of the target job in the list of jobs in the queue at the time of its entry sorted in increasing order of request sizes. $jobrank_{reqsize}$ is calculated by normalizing this position with respect to the total number of jobs in the queue.

Most of the existing strategies group jobs only in terms of request sizes, and find jobs with similar request sizes for predictions of queue waiting times. However, a single request size can correspond to widely varying queue waiting times as shown in Figure 1(a) that shows statistics for 1000 sample jobs from the CTC trace. For example, corresponding to the request size of 8 processors in the figure, we find that queue waiting times can vary from 1 minute to 32 hours. Hence, in addition to finding the similar jobs for a target job in terms of request sizes, we consider the rank of the job in the queue in terms of request size, thereby implicitly considering both the request size and the queue state for predictions. We consider target jobs with $jobrank_{reqsize}$ values less than a threshold, $threshold_{reqsize}$, as candidate quick starters. For fixing $threshold_{reqsize}$, we consider jobs in the near-term history, find two thresholds: $threshold_{reqsize1}$ as the maximum of $jobrank_{reqsize}$ values of the jobs with queue waiting times less than or equal to 1 hour, and $threshold_{reqsize2}$ as the minimum of $jobrank_{reqsize}$ values of the jobs with queue waiting times greater than 1 hour, and use the minimum of the two thresholds for $threshold_{reqsize}$. By using near-term history to find thresholds, PQStar uses the most recent history and hence also takes into consideration only those jobs having similar processor occupancy.

ERT Criterion: Similar to the request size criterion, we also use estimated run time (ert) criterion for identification of the target job as a quick starter. Specifically, PQStar finds a metric, $jobrank_{ert}$, using the list of jobs in the queue, at the time of entry of the target job, sorted in the ascending order of estimated run times, and finding the normalized position of the target job in the list with respect to the total number of jobs in the queue. PQStar marks the target job as a quick starter if its $jobrank_{ert}$ is less than a threshold, $threshold_{ert}$. $threshold_{ert}$ is found similarly to $threshold_{reqsize}$ by using the queue waiting times of jobs



(a) Wait Time Ranges for Different Request Sizes



(b) Wait time ranges for different ERTs

Fig. 1. Wait Time Ranges of Jobs for different Request Sizes and ERTs (CTC Trace)

in the near-term history. Figure 1(b) considers only the ERTs and their impact on queue waiting times. Similar to considering only request sizes, we find that a single ERT can correspond to wide variation of queue waiting times. Thus the existing strategies that use only ERTs to find similar jobs for predictions can give high upper bound values for predictions. By considering $jobrank_{ert}$, PQStar defines similarity using both the job and the queue state characteristics. The assumption behind the request size and the ERT criterion is that target jobs with small request sizes or ERT relative to the other jobs in the queue have higher chances of backfilling and hence can be quick starters.

Free Node Criterion The final criterion we use for mid-term history is based on the number of free nodes available to accommodate the target job. Thus this criterion explicitly takes into account the processor occupancy state in addition to the queue waiting state for identifying quick starters. Specifically, at the time of submission of the target job, PQStar finds the difference between the total number of free nodes available and the number of nodes requested by the jobs in the queue that have smaller request sizes or smaller estimated run times than the target job. If this difference is larger than the number of nodes requested by the target job, PQStar marks the target job as a quick starter. The assumption behind this criterion is that jobs in the queue with smaller request sizes or estimated run times have higher chances of backfilling and hence start earlier than the target job, thereby consuming some subset of free nodes.

For predictions with mid-term history, PQStar marks a target job as a quick starter if it meets any one of the three criteria, namely, request size, ert, or free node criteria.

3.3 Predictions using Far-term History

For those jobs for which mid-term history also cannot be used due to the above mentioned criteria not being met, we use the far-term history, which is the rest of the jobs in the history beyond the mid-term boundary. Among these far-term history jobs, PQStar extracts a subset of jobs with similar characteristics in terms of processor request sizes and estimated run time, and which have started executions. If all the jobs in this subset have queue waiting times of less than one hour, then it indicates that the target job will most likely have a wait time of less than or equal to one hour. Hence PQStar marks the target job as a quick starter.

3.4 Overall Predictions and Using IBL

In summary, PQStar tries to find similar jobs in the near, mid or far-term history, and uses a set of criteria to mark a job as a quick starter. In addition to considering only the job characteristics of request sizes and estimated run times, PQStar explicitly or implicitly considers the system states, namely processor occupancy and queue states, for defining similarity and for obtaining predictions.

In our evaluations we found that the near-term history typically consisted of about 50 jobs spanning around 1-3 hours and the mid-term history typically consisted of more than 500 jobs spanning around 10-25 hours. For target jobs that are either not marked as quick starters or for which similar jobs cannot be found in the near, mid or far-term history, PQStar uses an existing strategy for predicting queue waiting times. We use the IBL method by Li et al. [7] for these predictions, since we found in our experiments that IBL gives better predictions than QBETS [14]. IBL (Instance Based Learning) uses weighted sum of Heterogeneous Euclidean-Overlap Distances between job attributes to calculate similarities of jobs, and use similar jobs in the history to give point predictions for the target jobs based on 1-nearest neighbor or weighted average of k-nearest neighbors methods.

The entire algorithm followed in PQStar is illustrated in the flowchart shown in Figure 2.

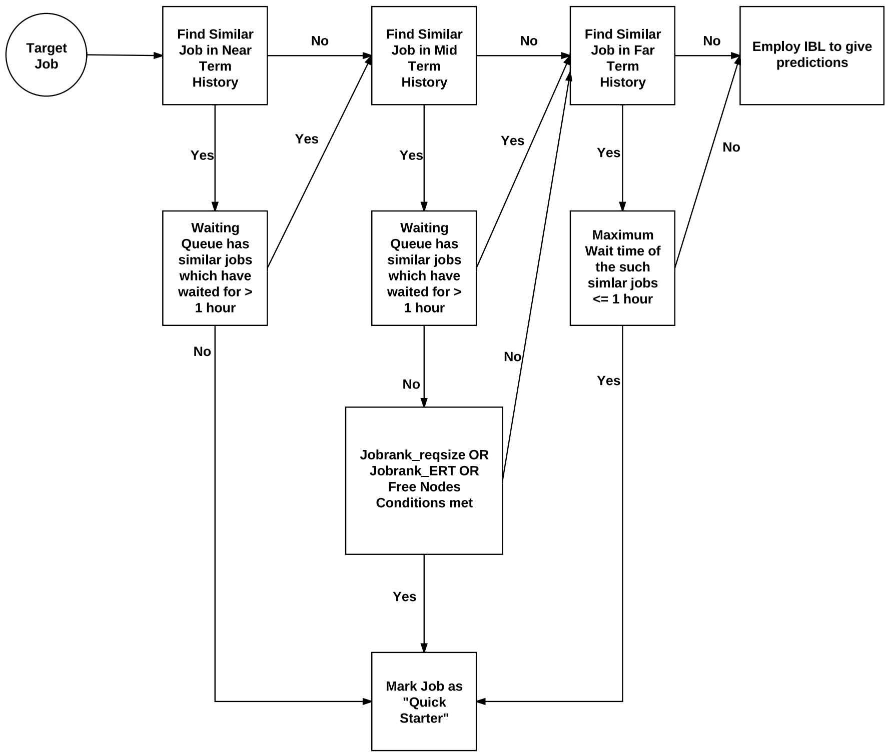

Target Job
Find Similar Job in Near Term History
No
Find Similar Job in Mid Term History
No
Find Similar Job in Far Term History
No
Employ IBL to give predictions
Yes
Yes
Yes
Yes
Yes
No
Waiting Queue has similar jobs which have waited for > 1 hour
Waiting Queue has similar jobs which have waited for > 1 hour
Maximum Wait time of the such simlar jobs <= 1 hour
No
No
No
Yes
Jobrank_reqsize OR Jobrank_ERT OR Free Nodes Conditions met
Yes
Mark Job as "Quick Starter"

Fig. 2. PQStar Methodology

4 Experiments and Results

4.1 Experimental Setup

The experiments were conducted using a discrete event simulator that we have developed. It creates a simulated environment of the jobs waiting in the queue and running on the system at different points of time. It is important to note that the simulator will only be keeping track of the jobs submitted to the system, and maintain their attributes including arrival times, wait times, actual runtimes and request sizes. It will not be simulating the actual scheduling algorithm used, thus avoiding assumptions about the underlying scheduling algorithm.

The simulator can be operated in two modes. In the first mode, the user can invoke the simulator with a supercomputing job trace/log in the Standard Workload Format (SWF) [17] as input, and obtain predicted queue waiting time of a new job. This mode is followed in the QBETS prediction system [14]. In this mode, the simulator creates the simulated environment of jobs in the system using the statistics available in the log. In the second mode, the simulator can be executed on the front end node of a batch system for which predictions are required. It will then track the arrivals, executions and exit of the real jobs submitted to the system, and will create the simulated environment using these real jobs. In this second mode, the job attributes maintained and used by the simulator can be obtained by queue and job management commands. For example on PBS based batch systems, the *qstat* command (with $-f$ option) will give all of the job parameters required by PQStar. Thus in the second mode, the predictions are obtained "live" at real time. The simulator is triggered by three primary events corresponding to job arrival, job beginning to execute and job termination. Whenever a job arrives, it is added to a waiting queue maintained by the simulator. As soon as a job's wait time is over and it starts executing, it is removed from the waiting queue and added to a running list in the simulator. Also at this time, the free nodes available in the system is decremented by the value equal to the job's request size. Once a job which is running completes its execution, it is removed from the running list and the free nodes available in the system is incremented by the value equal to the job's request size. This process is repeated for each job and thus a simulated system state is created using which we extract the processor state and the queue properties that are needed for our algorithm.

For each supercomputing trace in our experiments, we performed predictions for all the jobs starting from the 10001^{th} job up to a maximum of 50000 jobs or the end of the log. Each of the jobs in this set constitutes the evaluation data for which predictions were made. For a given target job for which waiting time is predicted, all the jobs submitted prior to it constitute the history. Out of these history jobs, we use a subset of jobs and use their waiting times for predicting for the target job. The subset is formed based on similarity to the target job and using near, mid and far term boundaries as described earlier. Once the target jobs start their execution and their wait times are known, they are added to the set of history jobs. We compared the predictions of our PQStar method with the results of IBL, QBETS and a parametric model, namely, using *log-uniform* distribution of the waiting times for predictions. We used the log-uniform model since it was found to give competitive results with QBETS in some cases [14].

4.2 Results

Predictions for Quick Starters We first show the effectiveness of using near-term history in PQStar for predicting quick starters. Table 2 shows the percentage of quick starters identified using near-term history and also the average number of Jobs in the Near Boundary. The results show that predictions based on near-term history contribute significantly to the identification of large number of quick starters.

Table 2. Percent of Quick starters marked using Near Boundaries in PQStar

Logs	**% Quick starters marked using Near Boundary**	**Average number of Jobs in the Near Boundary**
CTC	53	53
ANL	17	48
LANL	39	39
HPC2N	62	54
SDSC Paragon	61	43
SDSC Blue	51	46
SDSC SP2	50	45
DAS	66	59

Table 3 shows the percentage of the quick starters, that are successfully identified by log-uniform, QBETS, IBL and our method, PQStar. Successful identification corresponds to estimating the upper bound of the quick starters as less than or equal to one hour. We can see that our method, PQStar, performs better than the next best strategy, IBL, by successfully identifying up to 1.95 times more quick starters. It also successfully identifies up to 20 times more quick starters than QBETS. Our method is also more consistent and identifies more than about 80% of the quick starters irrespective of the log.

Misguiding Predictions These results show that our PQstar prediction system is highly successful in obtaining large number of *true positives*, i.e., identifying large number of quick starters. Our other objective is to minimize the number of *false positives*, i.e., number of jobs with long queue waiting times falsely identified as quick starters. We refer to these predictions as *misguiding predictions*. Table 4 shows that for the supercomputing traces used in our experiments, PQStar incurs such misguiding predictions for only 1-10% of the total number of jobs. The last column of the table also shows that 32-72% of the

Table 3. % of the QuickStarters Successfully Marked

Logs	**Log-Uniform**	**QBETS**	**IBL**	**PQStar**
CTC	2	5	43	84
ANL	11	42	61	78
LANL	8	67	85	88
HPC2N	3	19	49	80
SDSC Paragon	6	48	81	89
SDSC Blue	8	45	63	89
SDSC SP2	1	4	49	79
DAS	71	95	96	98

those misguided jobs have actual wait times of less than 4 hours. This indicates that for half of the misguiding predictions, the amount of misguidance is within reasonable limits.

Table 4. Misguiding Predictions

Logs	**% of Total Jobs Corresponding to Misguiding Predictions**	**% of Misguiding Predictions with Actual Wait Times of Less than 4 Hours**
CTC	9	59
ANL	4	72
LANL	1	61
HPC2N	10	46
SDSC Paragon	1	45
SDSC Blue	6	57
SDSC SP2	6	57
DAS	0.25	32

Overall Predictions Since IBL was found to be a better strategy than QBETS as shown in Table 3, our PQStar system uses IBL for predictions of non quick starters. We illustrate the comparisons of predictions of QBETS, IBL and PQStar in Figure 3, for 5000 jobs of CTC trace. Figure 3(a) shows the mean prediction error (absolute difference in predicted and actual wait times) for the different ranges of the actual wait times. From this, we can see that PQStar gives the least prediction error for the quick starters, and gives the same prediction error as IBL for the rest of the jobs, since PQStar uses IBL for jobs predicted as non-quick starters. In order to further elaborate the advantage of PQStar over IBL,we show a scatter plot of actual v/s predicted wait times for quick starters, as shown in Figures 3(b) and 3(c). We can clearly see that PQStar provides tight upper bounds, while IBL provides high upper bounds for a large number of quick starters.

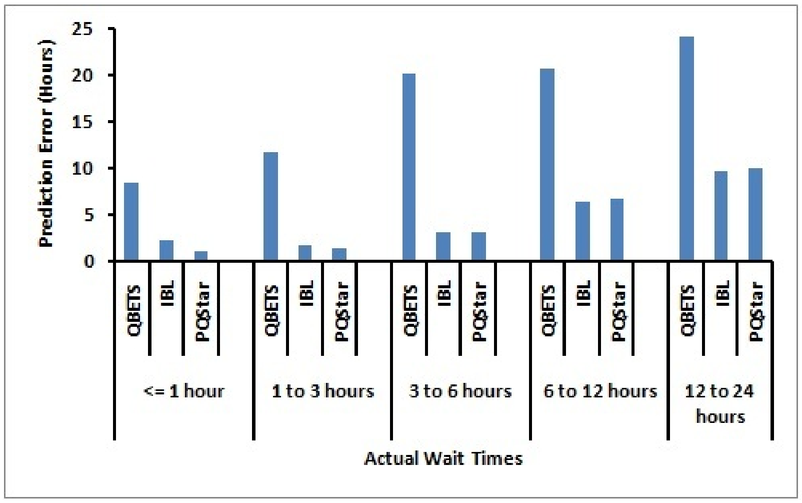
Figure 4 shows the distribution of predicted waiting times for different ranges of actual waiting times for all the jobs in the ANL (Figure 4(a)) and CTC (Figure 4(b)) traces for QBETS, IBL and PQStar. The graphs show that for jobs with actual waiting times of less than or equal to 1 hour, i.e., quick starters, PQStar is able to identify most of them as quick starters. For the other quick starters, the predictions by PQStar correspond to low ranges of queue waiting times. With QBETS and IBL, high predicted ranges are provided for a large number of these quick starters. For jobs with actual quick waiting times of 1 to 12 hours, PQStar gives smaller ranges of predicted queue waiting time for more jobs than QBETS and IBL.

The prediction time per job in our PQStar method is under a second and the total time take for the simulation to run for entire datasets was almost similar to that of both IBL and QBETS. Hence, there is minimal or no overhead added in terms of prediction time by PQStar to the existing IBL method.

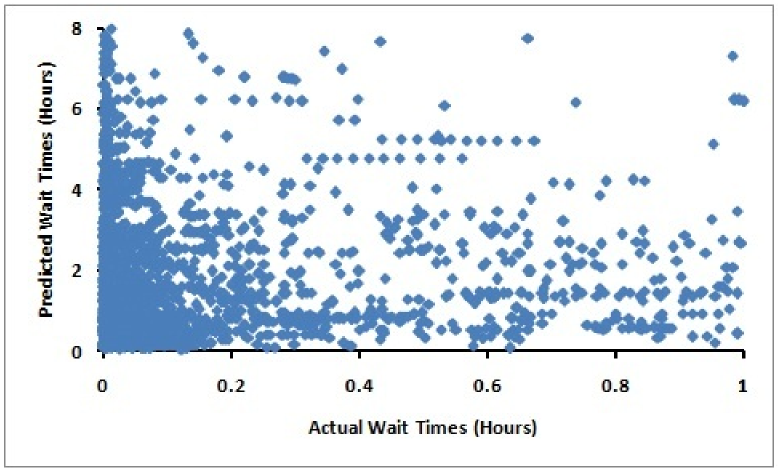
RMS Error and Response Time Predictions In order to evaluate the effect of the predictions of quick starters by PQStar on the overall accuracy of predictions, we calculate the RMS (Root Mean Square) value between the actual and predicted queue waiting times for all the jobs. We compute the percentage decrease in RMS error for PQStar from the RMS errors of the other methods. For example, for comparison of RMS errors of PQStar and QBETS we compute $\frac{rmserror_{qbets} - rmserror_{pqstar}}{rmserror_{qbets}}$.

We denote the percentage decrease as $rmsdec_{fromlu}$, $rmsdec_{fromqbets}$ and $rmsdec_{fromibl}$, for comparisons with log-uniform, QBETS and IBL, respectively. Positive values for the percentage decrease indicate better predictions by PQStar. Table 5 shows the decrease in RMS error due to PQStar for all predictions. We find that PQStar gives better prediction accuracy than the other methods. Our method results in up to 90% average improvement in overall prediction accuracy of all the jobs over log-uniform and up to 64% average improvement over QBETS predictions. The average improvement due to PQStar is only about 2% when compared to IBL since PQStar uses IBL for predictions of non quick-starters. The actual waiting times and the prediction errors for these non quick-starters have large values, and these large prediction errors dominate the overall RMS error considering all the jobs. Hence the difference in RMS error between PQStar and IBL is small. The last column of the table shows $nrmsdec_{fromibl}$, the percentage decrease in normalized RMS error due to PQStar when compared to IBL. The normalized RMS errors are calculated by normalizing the individual prediction errors using the actual waiting times. As the results in this column show, PQStar results in significant gain up to 42% in overall prediction accuracy when compared to IBL.

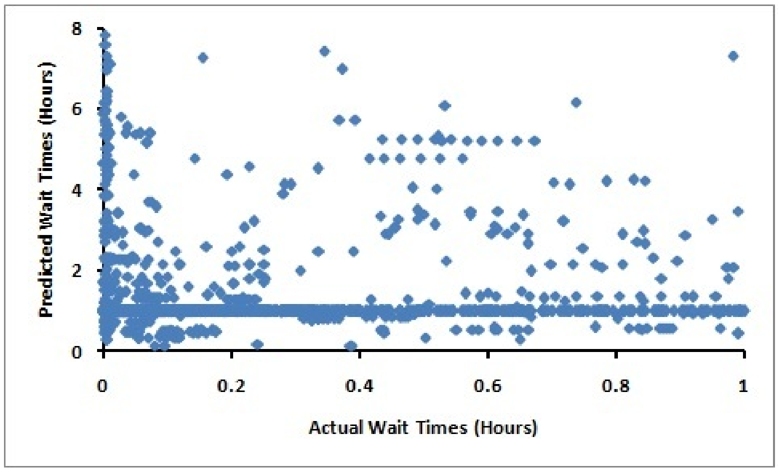
We also compute the percentage difference in predicted and actual response times for each job, where response time is the sum of queue waiting time and



(a) Prediction Error for Different Actual Queue Waiting Time for QBETS, IBL and PQStar

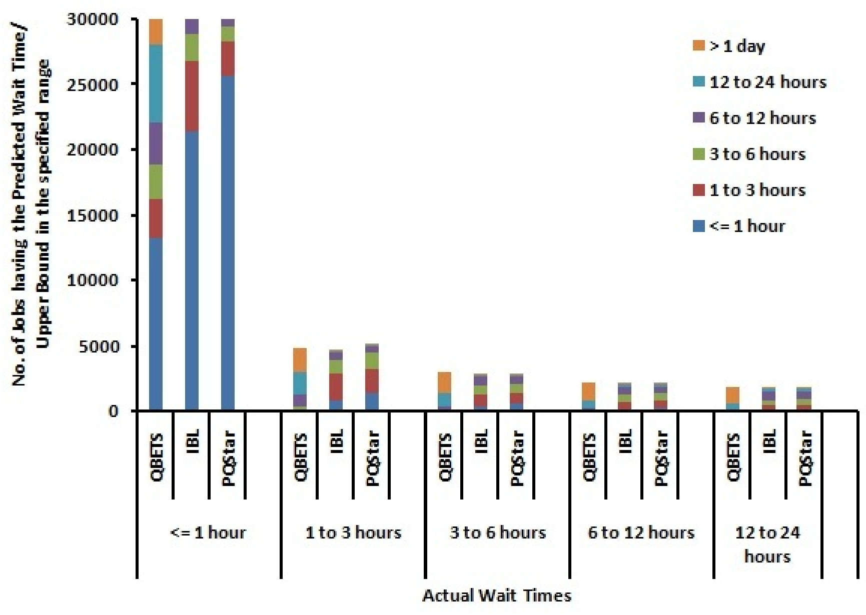


(b) Actual Wait Time v/s Predicted Wait Time for IBL

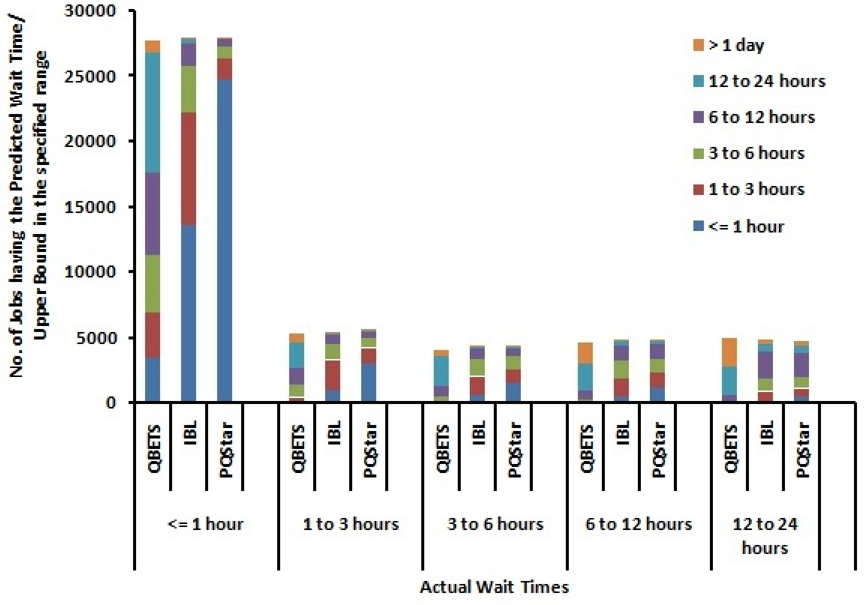


(c) Actual Wait Time v/s Predicted Wait Time for PQStar

Fig. 3. Prediction Comparison for QBETS, IBL and PQStar using 5000 jobs of CTC trace



(a) ANL Trace



(b) CTC Trace

Fig. 4. Distributions of Predicted Queue Waiting Times for Different Actual Queue Waiting Time for QBETS, IBL and PQStar using ANL and CTC Traces

Table 5. Prediction Accuracy: RMS Errors

Logs	**rmsdec$_{\textbf{fromlu}}$**	**rmsdec$_{\textbf{fromqbets}}$**	**rmsdec$_{\textbf{fromibl}}$**	**nrmsdec$_{\textbf{fromibl}}$**
CTC	80	58	2	42
ANL	75	61	2	22
LANL	90	57	1	5
HPC2N	84	59	2.25	42
SDSC Paragon	79	64	1.5	6
SDSC Blue	77	58	2	33
SDSC SP2	71	56	1.75	38
DAS	19	7	0	0

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 $PPE_{rt} = \frac{|predictedwaitingtime - actualwaitingtime|}{actualresponsetime}$. This metric determines the amount of impact of the prediction errors on jobs of different lengths or execution times. A prediction in queue waiting time with an error of 1 hour will have higher impact on a job whose execution time is 15 minutes than for a job whose execution time is 2 days.

Further, to define good predictions, we divide the waiting and run times into different *bins*. Each slab represents a range of wait/run times and is associated with an interval. The wait/run time of a job is rounded off to the nearest interval values associated with the slab to which the wait/run time belongs. Table 6 refers to the bins and interval size for each slab used for our experiments. For example, if the wait time of a job is 37 minutes, the interval size that will be used is 15 minutes (first row). Hence the wait time is rounded off to 45 minutes, which is the nearest next 15 minute interval. As can be seen, the idea of using bins is to give different tolerance limits in prediction errors for different predicted and actual wait times. Prediction error of 15 minutes is large for a job whose actual waiting time is 20 minutes, while it is small for a job whose actual waiting time is 2.5 days.

Table 6. Bins and the corresponding Intervals

Bins	**Intervals**
less than or equal to 1 hour	15 min
1 hour to 3 hours	30 min
3 to 6 hours	1 hour
6 to 12 hours	3 hours
12 to 24 hours	6 hours
1 day to 2 days	12 hours
greater than 2 days	24 hours

We consider a prediction for a job as a *good prediction* if the rounded values of actual and predicted queue waiting times lie in the same slab or if its PPE_{rt} value is within 10%. Tables 7 shows the average PPE_{rt} values and percentage good predictions obtained by the various methods. The table shows that the average PPE_{rt} is up to 35 times less and the number of good predictions is up to 58% more with PQStar when compared to the other methods.

Table 7. Prediction Accuracy: % difference in Predicted and Actual Response Times of Different Prediction Methods

Logs	**Log-Uniform**		**QBETS**		**IBL**		**PQStar**	
	Average $\mathbf{PPE_{rt}}$	**% Good predictions**	**Average $\mathbf{PPE_{rt}}$**	**% Good predictions**	**Average $\mathbf{PPE_{rt}}$**	**% Good predictions**	**Average $\mathbf{PPE_{rt}}$**	**% Good predictions**
CTC	41.1	3	19.6	6	1.4	39	0.5	60
ANL	20.7	5	17.5	32	2.21	36	1.4	57
LANL	35.2	9	22.4	61	0.12	91	0.06	94
HPC2N	44.7	5	16.5	17	1.96	46	0.68	66
SDSC Paragon	45.7	6	18.3	45	0.45	79	0.24	85
SDSC Blue	31.7	8	13.1	35	2.01	53	0.75	70
SDSC SP2	55.2	3	32.3	6	2.86	45	1.2	64
DAS	2.6	73	0.25	96	0.01	98	0.007	99

Thresholds for Quick Starters For all the above results, we have used a queue waiting time threshold of 1 hour for the definition of quick starters. Jobs with actual queue waiting times less than this threshold are marked as quick starters. This threshold is based on the assumption that waiting time of less than one hour may be short and prediction errors up to that limit may be acceptable for the user. We used the value of one hour as a threshold to target jobs with queue waiting times less than this threshold to improve/tighten the bounds of these jobs, since this class of jobs form a majority of the jobs as we had shown in Table 1. However, there have been a series of works [18] [19] which show that the response time of a job should be less than 20 minutes to consider a job submission session as interactive. In this experiment, we analyse the effect of the changing thresholds for the quick starters by using different thresholds in PQStar for predicting quick starters. Table 8 shows the impact of changing the thresholds for quick starters from 10 minutes to 1 hour on the PQStar predictions of quick starters for the CTC trace. We find that PQStar consistently identifies more than 80% quick starters for all the thresholds, and the variation in threshold does not have an impact on the predictions of quick starters.

Table 8. Impact of changing the thresholds for quick starters (CTC Trace)

Quick Starter Threshold (in minutes)	**% of Quick Starters correctly identified**
10	84.22
20	83.99
30	83.74
40	83.69
50	83.62
60	84.28

In summary, PQStar performs better than both IBL and QBETS, and it also outperforms the parametric model, using log-uniform distribution, in all the above shown aspects. From these results, we can clearly see that our method is providing much more aggressive bounds for the quick starters compared to rest of the methods, and also the under predictions is kept to limited amounts.

5 Conclusions and Future Work

In this work, we had developed a prediction system called PQStar for identification of quick starters or jobs whose actual queue waiting times are less than or equal to 1 hour. These quick starters form a majority of the job submissions in many supercomputer traces. In this work we consider both job characteristics, namely, request size and estimated runtime time, and the state of the system, namely the queue and processor occupancy states, for predictions. By means of experiments with different supercomputer traces, we showed that that our prediction strategies can lead to correct identification of up to 20 times more quick starters and provide tighter bounds for these jobs, and thus result in up to 64% higher overall prediction accuracy than existing methods.

We currently use the IBL method for predictions of jobs with potential long queue waiting times. We plan to explore alternate strategies for predictions of such jobs. We also plan to develop techniques for predictions of execution time in order to predict total response times. Predicting execution times for jobs submitted to batch systems is challenging due to limited history. We finally plan to build scheduling and metascheduling strategies that use these stochastic predictions to select the appropriate resources for job executions.

References

1. IBM Load Leveler. [Online]. Available: http://www.redbooks.ibm.com/abstracts/sg246038.html

2. PBS Works. [Online]. Available: http://www.pbsworks.com/
3. Platform LSF. [Online]. Available: http://www.platform.com/workload-management/high-performance-computing
4. MAUI Scheduler. [Online]. Available: http://www.supercluster.org
5. Tera Grid Karnak Prediction Service. [Online]. Available: http://karnak.teragrid.org/karnak/index.html
6. Parallel Workload Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html
7. H. Li, D. L. Groep, and L. Wolters, "Efficient Response Time Predictions by Exploiting Application and Resource State Similarities," in *GRID '05 Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, 2005, pp. 234–241.
8. H. Li, J. Chen, Y. Tao, D. L. Groep, and L. Wolters, "Improving a Local Learning Technique for Queue Wait Time Predictions," in *CCGRID '06 Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006, pp. 335–342.
9. W. Smith, I. T. Foster, and V. E. Taylor, "Predicting Application Run Times Using Historical Information," in *IPPS/SPDP '98 Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 122–142.
10. W. Smith, V. E. Taylor, and I. T. Foster, "Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance," in *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, 1999, pp. 202–219.
11. D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Scheduling - A Status Report," in *Proceeding in JSSPP'07 Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, 2004, pp. 1–16.
12. H. Li, D. Groep, J. Templon, and L. Wolters, "Predicting Job Start Times on Clusters," in *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, 2004.
13. A. B. Downey, "Predicting Queue Times on Space-Sharing Parallel Computers," in *IPPS '97 Proceedings of the 11th International Symposium on Parallel Processing*, 1997, pp. 209–218.
14. D. Nurmi, J. Brevik, and R. Wolski, "QBETS: Queue Bounds Estimation from Time Series," in *Proceeding in JSSPP'07 Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, 2007, pp. 76–101.
15. J. Brevik, D. Nurmi, and R. Wolski, "Predicting Bounds on Queuing Delay for Batch-Scheduled Parallel Machines," in *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2006, pp. 110–118.
16. D. N. J. Brevik and R. Wolski, "Using Model-Based Clustering to Improve Predictions for Queueing Delay on Parallel Machines," pp. 21–46.
17. Standard Workload Format. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html
18. E. Shmueli and D. G. Feitelson, "Uncovering the Effect of System Performance on User Behavior from Traces of Parallel Systems," in *MASCOTS '07 Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2007, pp. 274–280.
19. J. Zilber, O. Amit, and D. Talby, "What is worth learning from Parallel Workloads?: A User and Session based Analysis," in *ICS '05 Proceedings of the 19th annual international conference on Supercomputing*, 2005, pp. 377–386.