Machine learning based queuing time prediction of batch scheduler on supercomputers

Bowen Zhang, Xingyu Chen, Zengyi Wang

Southern University of Science and Technology
Department of Computer Science and Engineering

Abstract—To share high performance computing resources, HPC clusters queue jobs to provide computing services. Due to the limited computing resources, naturally there is the problem of queuing. The prediction of queuing time can improve the resource utilization of a HPC cluster. There are some elastic jobs which can be run on any number of nodes in parallel and there are frameworks (e.g., parsl) that enable jobs to be executed in parallel on multiple nodes. Knowing the exact queue time is important for these jobs in order to minimize the response time. Response time is queue time plus running time. Our study will explore machine learning models that can accurately predict queue time, and propose an architecture that can improve model accuracy by using similarity calculation. We'll also talk about what we're doing now.

1. Introduction

1.1. Supercomputers

Nowadays, with the increasing demand for computational resources in the field of basic science, people tend to use computers with extreme computational power to process programs, and such computers with extreme computational power are called supercomputers. Supercomputers contain thousands of computing resources (CPUs and GPUs), and users request the appropriate number of computing resources to process tasks for them according to their needs. Upon receiving resource requests from different users, the supercomputer runs a program called a scheduler to allocate the computing resources. In other words, the supercomputing center provides a shared pool of resources, each task occupies part of the resources when it is executed, and multiple tasks are scheduled by the scheduler to allocate computing resources according to certain rules in a queue.

1.2. A scheduling technique to improve resource utilization

There is a scheduling technique called backfill, which improves the resource utilization of the system. We define tasks that consume more computational resources as large tasks and tasks that consume fewer computational resources as small tasks. Backfill works by reserving resources for the execution of large tasks. At the same time, the time gaps generated during the execution of large tasks are used to prioritize the small tasks in the waiting queue so that they are executed before the large tasks. Reserving resources for tasks with more computational resources avoids long waiting time for large jobs, and prioritizing small jobs in the queue improves the response time of small tasks, both of which increase the number of working nodes and improve the resource utilization of the system.

1.3. Motivation

Overall, predicting the queuing time submitted to a supercomputer processing system is both important for users to schedule their jobs and can help the scheduler make informed scheduling decisions.

Users would gain many benefits if they could predict the queuing time for jobs on a batch processing system. First, the predicted time can help the user plan to manage its work and help the user try to avoid not being able to complete the work by the deadline. When the queue prediction time is known to be too long, the user can choose another queue, and this practice can also reduce the load on certain queues of the computer and make the load more balanced.

In addition, there is a class of jobs that can be executed in parallel on an arbitrary number of nodes, and there are frameworks (e.g., parsl [1]) that enable job to be executed in parallel on multiple nodes. For this class of jobs, it is important to know the waiting time for different number of node requests, which will determine how many nodes can be used to execute the job in parallel to achieve the shortest response time. Also, the scheduler can effectively use this prediction to make scheduling decisions and select the appropriate number of computational resources and queues for each computational job to improve the utilization of computational resources.

1.4. Research Objectives

Some previous papers have shown that it is possible to predict a bound of the queuing time of batch scheduled jobs. Today, machine learning makes it possible to analyze and predict large amounts of data more quickly and accurately. With the recent advance of machine learning methods, we will revisit this problem and see if we could use machine learning to predict the queuing time of batch schedulers more accurately. We will explore different machine learning methods in this project to perform different levels of predictions.

1.5. Research Challenges

First, it is impractical to obtain or infer the priority and scheduling algorithms of the jobs because most of the scheduling algorithms are not open source. It is difficult for us to know the exact workflow of most scheduling algorithms, adding difficulties to predicting queueing times.

In addition, predicting the queue wait time faces the difficulty that the wait time of a particular task depends partly on the future task arrivals and the execution of tasks at each compute node, which are unknowable at the time of prediction.

Also, the system has a backfill mechanism that causes a portion of small tasks to be executed earlier. This makes it more difficult to predict the waiting time of the queue.

2. Related work

The existing papers provide different prediction methods from different machine learning models. These methods can be broadly classified into two categories: the first category will calculate the similarity between tasks and group the tasks with similarity into one category. The execution time of that task is then predicted based on the execution time of similar tasks. The second category is to directly use a machine learning model to predict the execution time of a task.

The first class of methods is the most common in current research.W. Smith used this class of methods in [2] where summary statistics about the state of resources (e.g., number of running jobs and idle cpu) are used as attributes. In other work by W. Smith [3] [4], runtime predictions are derived using historically similar runs, and these estimates are further used to simulate scheduling algorithms such as FCFS, LWF. Hui Li's [5] approach is to first classify tasks by similarity and then search them using a genetic algorithm to keep the relative prediction error between 0.35 - 0.70.

Unlike computing similarity, there are also studies that directly pass the data into a machine learning model directly to predict the queuing time. In a recent paper by Ju-Won Park [6], it used the HMM approach to improve the prediction accuracy of traditional algorithms by 60 percents. In Rajath Kumar's [7] work, he first predicts the waiting time of jobs using a dynamic k-nearest neighbor (kNN) approach. Then multiclass classification of all classes of jobs is performed using support vector machines. The probabilities obtained using the above two methods were used to provide a set of predicted waiting time ranges with probabilities. The scheduling policy designed based on this predicted time range reduced the average queuing wait time of jobs by 47 percents.

From the point of practical application, the result of the present work is still not very well. The error of categorical prediction or direct prediction is very large. For practical application, too large error is not acceptable. Our research is aimed at further improving the accuracy of predictions and making practical applications possible.

3. Data Analysis

In this section, we mainly focus on analyzing existing dataset and try to extract key information. Now we have completed the pre-processing and data analysis of theta cluster data.

There are many factors that effect queue time. It can be roughly divided into queue state, system state and job state. I will analyze theta data from these three directions. For the generalization of the model, we only analyze the default queue. First, we defined the label and counted the number of different labels in Figure 1.

Label	Meaning
1	less than one hour
2	1 hour to 3 hours
3	3 hours to 6 hours
4	6 hours to 12 hours
5	12 hours to 24 hours
6	greater than one day

We find this distribution of Theta data odd, since jobs whose queue time are less than one hour usually account for more than 50% of the total number of jobs in a cluster [8], perhaps because theta smaller tasks typically use backfill queue.

3.1. Job State

Obviously, queue time has a certain relationship with request node. We do data analysis according to the request of each node number. Another factor that is obviously related to queue time is request run time. We will analyze the relationship between request node in Figure 3. and queue time and between request time in Figure 2. and queue time.

From the analysis, when request time is less than 40000s, queue time increases with request time. However, when the request time is greater than 40000s,

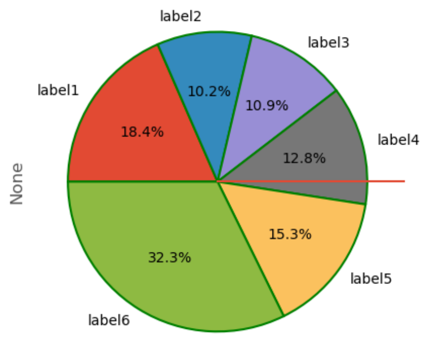

Figure 1. The distribution of label

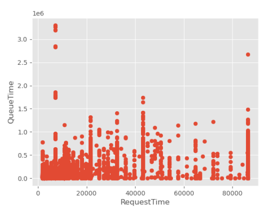

Figure 2. scatter diagram request time with queue time

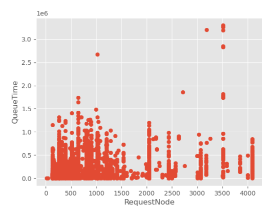

Figure 3. scatter diagram request node with queue time

Figure 2 distribution is very disorganized and inconsistent with our expectations. In the scatter diagram of request Figure 3 node and queue time, when the nodes are large, the distribution of queue time is also very scattered, generally low, with many outliers. The distribution of queue time when the request node is large does not conform to our expectations. When the request node is small, the distribution is more concentrated with fewer outliers, which is more consistent with our expectations. The more resources applied by jobs, the more inconsistent with our intuitive prediction, and the more outliers, which may be due to the submitter notifying the supercomputer cluster manager in advance, or the account has priority. All these factors make it difficult for us to predict queue time using machine learning.

According to the above data analysis, we put forward a point of view: Jobs with larger resource scale are more influenced by human factors. To verify the above conjecture, we analyze the distribution of request time and request node Figure 4, and use request time * request node to measure the application of system resources for a jobs. Then we make a scatter diagram request time * request node with queue time

According to data analysis Figure 4, when request node is less than 1000, the distribution of request time increases significantly in the area with large value with the increase of request node. When the request node is

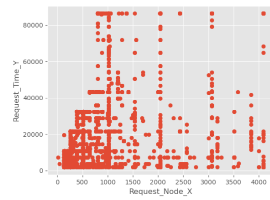

Figure 4. scatter diagram request time with request node

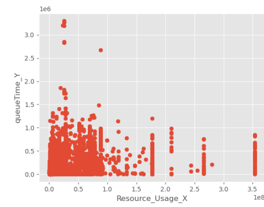

Figure 5. scatter diagram Resource with queue time

near 1000, the request time is evenly distributed. Subsequently, as the request node grows, the request time is more concentrated and distributed in the low/middle area. As can be seen from Figure 5, when the resource application of a Job is significantly larger than that of other Jobs, its waiting time will decrease significantly. This also proves the above conjecture.

3.2. System/Queue State

We use default queue as our dataset. Hardware resources corresponding to the default queue are not isolated. Therefore, the system state affects the queue time of the job to some extent. And there is no doubt that the state of the queue when a job is submitted affects the queue time of the job. Since we have not reached an agreement on the features of system state and queue state at present, I only use some features 3.2 that I think are very typical here.

We defines two features for system state and queue state.

name	Meaning
prpjobRankReqsize	The position of the target job in the list of running jobs in the system at the time of its entry sorted in increasing order of request sizes
queueJobRankByReqsize	The position of the target job in the list of queue jobs in the default queue at the time of its entry sorted in increasing order of request sizes

The reason why I use rank instead of the number of node applications is that for queue job list, the smaller the application node, the more likely the job will be backfilled first. The size is relative, so I use rank. For running job list, The size of a running job is also relative. A smaller job is more likely to be executed. After a running job ends, a certain number of nodes will be released, and we compare the number of released nodes with the number of job applications. The job will be executed only when the number of nodes released exceeds the number of applications. Figure 6. shows relationships between prpjobRankReqsize and queue time and Figure 7. shows relationships between queueJobRankByReqsize and queue time.

The results of the data analysis showed that there was not much relationship between them. Even prpjobRankReqsize is very different from what we expected. I think the reason for this is that there are a large number of small jobs use backfill queue in the theta, while most of the tasks in the default queue are large

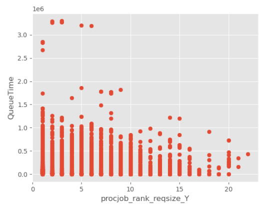

Figure 6. scatter diagram request node rank with queue time in running jobs list

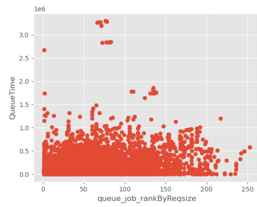

Figure 7. scatter diagram request node rank with queue time in queue jobs list

jobs and backfill has little influence. The features defined above are considered from the perspective of backfill. The allocation of queues like Theta is very unreasonable. Other supercomputer clusters do not have so many tasks to be executed on backfill queues. Subsequently, we will analyze other data sets.

4. Progress

In this section, we will discuss the current progress of our experiment. We first cluster the dataset and train a predictor for each cluster. When predicting a sample, we first judge its clustering category and predict it with the corresponding predictor. The details will be covered below.

4.1. Experimental Purpose

We continue to use the linear model based on our previous experiments and add several features to it. The reason why linear model is used for the experiment is that for large-scale data, its convergence speed is fast, which is convenient for us to test features. At the same time, in our previous planning, we considered the combination of linear model and LSTM. However, LSTM results should only be input as a feature into the linear model, and our linear model should be able to predict without LSTM. So, our first experiment purpose is to train a linear model that could make good predictions.

In our previous plan, we wanted to combine the similarity with the model. In a recent study, A. Pal [9] did something similar. He clustered the data sets and trained a predictor for each cluster. We used clustering on a linear model based on his idea. We use K-means for clustering. The role of clustering is actually to transform data from big heterogeneous pool to small homogeneous clusters. For dataset transformation, there are many similar studies [10] [11]. So, the purpose of our second experiment is to use K-means to improve the accuracy of the linear model.

Finally, our final research goal is to deploy the model to the SUSTECH Taiyi cluster. Therefore, we performed data mining and data cleaning on the data from Taiyi. The obtained data was pre-processed and trained on the outside model. So, the purpose of our third experiment is to train Taiyi data on our model.

4.2. Procedure

In this section, we will talk about our experimental procedure to achieve these purposes above. We mainly reproduced experiments in paper [9]. Section 4.2.1 describes the whole process, and the rest subsections explains details for each small part.

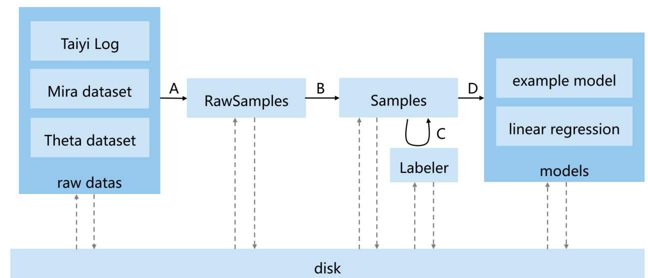

Figure 8. Experimental procedure

4.2.1. Structure. In paper [9], their model was constructed in the following way: First, they extracted four features from the dataset, which contains CPU-hours, number of CPUs, queue occupancy (number of jobs in the queue at the time of job submission), and system load (number of occupied nodes during submission). For training, they do K-means clustering first, and for each cluster, they generate different training model.

Fig 8. shows the whole structure of the experiment. First, we need to convert multiple raw data into a unified format for further process. In this case, procedure A describes the process that transfering different raw data into a RawSample format, which contains submit timestamp, start timestamp, finish timestamp, CPU node used, request wall time, and queue name. Then, RawSample need to be construct to the feature that we actually used. Procedure B will convert RawSample format into Sample format, which contains CPU hours, CPU number, queue load, system load, and actual queue time. Now data is clean for training. Procedure C will first do clustering as the original paper said to give each sample a label, then procedure D will use different model to train these labeled samples.

4.2.2. Preprocessing. Procedure A is about convert different raw datasets into RawSamples. We use three different dataset for our model: Theta, Mira, and Taiyi log. Theta and Mira are public dataset available on the Internet. We use about 70,000 and 150,000 records in them respectively. Taiyi log is from our institution HPC system. It contains about 3,000,000 records originally.

In the treating process for Taiyi, we noticed that most records did not contain wall time. We suspect that it is because different queues have time limit themselves, so most user will not worry about running their tasks forever and ignore setting wall time. This is an alarm because it means that the wall time feature may not be a good feature for all the high performance cluster. After filtering valid wall time and other features, only about 600,000 records remained.

4.2.3. Sample Converting. This part is to add timing information to the data, describing the state of the system and the queue when the Job is submitted, and also to standardize the format of the data. In fact, this part converts the RawSample format to Sample format and calculates the attributes system load and queue load that hold the timing information.

4.2.4. Labeling. In this section, we use the K-means algorithm to cluster the dataset and label each data with its category. In fact, the clustering is performed using the four features mentioned in the previous section. Also, this step saves the centroids of each cluster so that we can determine which cluster a new data belongs to.

4.2.5. Training and testing. Our training and prediction processes are combined with similarity methods, which here refer to clustering.

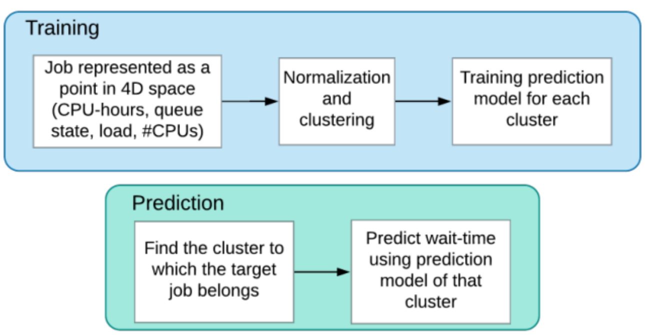

Figure 9. Experimental procedure

Fig 8. shows the process of our training and prediction. To perform training, we divide the dataset into multiple datasets by clustering. For each dataset, we train with a linear model. In testing, for each sample, we determine its clustering category by centroids and use the corresponding predictor to make predictions.

4.3. Result

In this experiment, we conducted experiments on three datasets. The first dataset is the Theta dataset from Argonne National Laboratory, USA, with a data volume of 53748. The second dataset is the Mira dataset, also from Argonne National Laboratory with a data volume of 157190. The third dataset is Taiyi dataset, from Taiyi Supercomputer of Southern University of Science and Technology, China, with data volume of 601500.

We use two evaluation metrics, the average absolute error (AAE) and the percentage prediction error (PPE) of response time, to measure the effectiveness of our work.

AAE is the average of the absolute difference between predicted and actual waiting times. According to R. Kumar and S. Vadhiyar's study [12], PPE is defined as

$$PPE = \frac{|predictedwaitingtime - actualwaitingtime|}{actualresponsetime}$$

In this formula, predictedwaitingtime refers to the predicted queuing time, actualwaitingtime refers to the actual waiting time, and actualresponsetime refers to the actual response time, which is the actual waiting time plus the actual execution time.

The metric PPE indicates the extent to which the prediction error affects the work for different execution times. The same queue waiting error has a greater impact on tasks with shorter execution times; for example, if the error in predicting queue waiting time is 1 hour, then this prediction has a higher impact on jobs with an execution time of 10 minutes than on jobs with an execution time of 10 hours. Thus, the metric PPE is a good reflection of the property of the degree of impact of prediction error on jobs with different execution times.

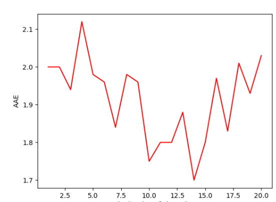

Figure 10. the line graph of the AAE values in the Theta dataset as the number of clusters k varies

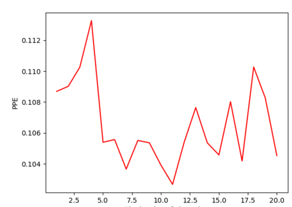

Figure 11. The line graph of the PPE values in the Theta dataset as the number of clusters k varies

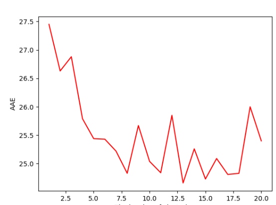

Figure 12. The line graph of the AAE values in the Mira dataset as the number of clusters k varies

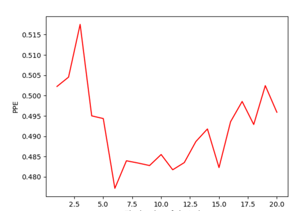

Figure 13. The line graph of the PPE values in the Mira dataset as the number of clusters k varies

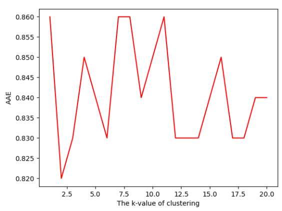

Figure 14. The line graph of the AAE values in the Taiyi dataset as the number of clusters k varies

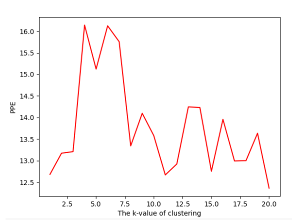

Figure 15. The line graph of the PPE values in the Taiyi dataset as the number of clusters k varies

We explored the effect of the change in the number of groups K of clusters on AAE and PPE. The above six line graphs are the results of the experiments, and each row represents the Theta dataset, the Mira dataset and the Taiyi dataset, respectively. The horizontal coordinates in the line graphs are the number of groupings K of the clusters, and the vertical coordinates are the AAE and PPE, separately.

4.4. Analysis and Conclusion

In this section, we will analyze the experimental results.

In the data sets theta and mira, PPE and AAE have a very significant change with increasing K values. For theta, the AAE reaches its lowest value at K = 14 and the PPE reaches its lowest value at K = 11. Compared to no clustering, the highest decrease in AAE is 15% and the highest decrease in PPE is 5.5%. For mira, the AAE reaches its lowest value at K = 13 and the PPE reaches its lowest value at K = 6. Compared to no clustering, the highest decrease in AAE is 10.1% and the highest decrease in PPE is 5.0%. For Taiyi's data, the effect of clustering on it is not obvious.

	PPEDecrease	AAEDecrease
theta	5.5%	15%
mira	5.0%	10.1%
Taiyi	2.5%	4.7%

The effect brought by clustering is not so obvious, especially on the clusters of Taiyi. At the same time, the PPE and AAE metrics to measure the goodness of the model are not always that reliable. Although the values of our model's indicators AAE and PPE are not bad, our models sometimes have prediction errors of up to tens of times.

And we have achieved results similar to the original paper. We compare the results of mira taking K=6 and theta taking K=10 with the results of the original paper, respectively.

Log name	PPE	AAE	paper PPE	paper AAE
theta	0.1	1.75	0.2	1.4
mira	0.47	25.43	0.55	21.78

The results show that our AAE, although slightly higher than the results of the original paper, is very close to the results of the original paper, while the PPE is already better than the results of the original paper. Even so, our model did not work well in real-world tests, even though we managed to replicate cutting-edge research results.

Based on the above experiments, we draw the following conclusions.

- Linear model predictions are feasible.
- clustering does not have a particularly significant effect on the improvement of model accuracy. However, the combination of similarity and model does improve the model accuracy.
- PPE and AAE are not a good indicator of how well the model actually performs.

5. Future Work

1) We need to find a more realistic evaluation indicator. Existing indicators such as PPE, AAE are difficult to reflect the actual use effect. Also, there is a need to have a common metric to measure the prediction results among different models, for example, some models are doing classification prediction and some are doing accurate prediction.
2) According to our experimental results, the practical use of the models from the cutting edge research is very poor. We need to find a model with practical value and combine it with temporal information (LSTM) and similarity algorithms.
3) Also, deploy our model to the school's Taiyi cluster. In the process of data processing in Taiyi, we found that a considerable part of Taiyi data is without the feature of Wall time, but the traditional model is highly dependent on the feature of Wall time. So we need to find a model that does not rely on the feature of Wall time.

References

[1] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster et al., "Parsl: Pervasive parallel programming in python," in Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, 2019, pp. 25–36.

[2] W. W. Smith, Resource management in metacomputing environments. Northwestern University, 1999.

[3] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 122–142.

[4] W. Smith, V. Taylor, and I. Foster, “Using run-time predictions to estimate queue wait times and improve scheduler performance,” in Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 202–219.

[5] H. Li, D. Groep, and L. Wolters, “Efficient response time predictions by exploiting application and resource state similarities,” in The 6th IEEE/ACM International Workshop on Grid Computing, 2005. IEEE, 2005, pp. 8–pp.

[6] J.-W. Park, M.-W. Kwon, and T. Hong, “Queue congestion prediction for large-scale high performance computing systems using a hidden markov model,” The Journal of Supercomputing, pp. 1–22, 2022.

[7] R. Kumar and S. Vadhiyar, “Prediction of queue waiting times for metascheduling on parallel batch systems,” in Workshop on Job Scheduling Strategies for Parallel Processing. Springer, 2014, pp. 108–128.

[8] Kumar, Rajath and Vadhiyar, Sathish, “Identifying quick starters: towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs,” in Workshop on Job Scheduling Strategies for Parallel Processing. Springer, 2012, pp. 196–215.

[9] A. Pal and P. Malakar, “An integrated job monitor, analyzer and predictor,” in 2021 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2021, pp. 609–617.

[10] P. Murali and S. Vadhiyar, “Qespera: an adaptive framework for prediction of queue waiting times in supercomputer systems,” Concurrency and Computation: Practice and Experience, vol. 28, no. 9, pp. 2685–2710, 2016.

[11] D. Nurmi, J. Brevik, and R. Wolski, “Qbets: Queue bounds estimation from time series,” in Workshop on Job Scheduling Strategies for Parallel Processing. Springer, 2007, pp. 76–101.

[12] R. Kumar and S. Vadhiyar, “Identifying quick starters: Towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs,” 01 2013.