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,

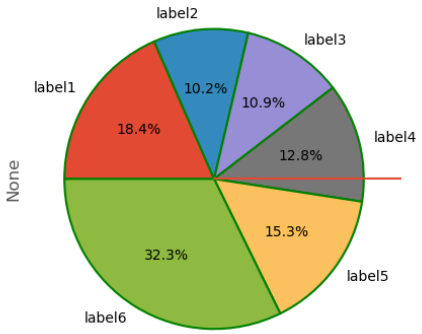

Figure 1. The distribution of label

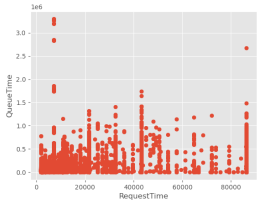

Figure 2. scatter diagram request time with queue time

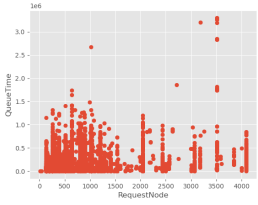

Figure 3. scatter diagram request node with queue time

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

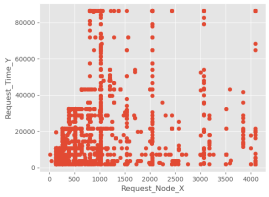

Figure 4. scatter diagram request time with request node

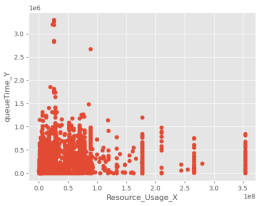

Figure 5. scatter diagram Resource with queue time

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

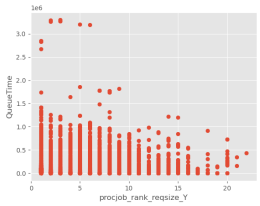

Figure 6. scatter diagram request node rank with queue time in running jobs list

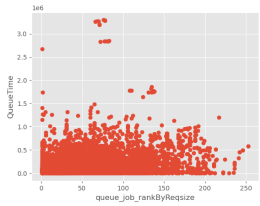

Figure 7. scatter diagram request node rank with queue time in queue jobs list

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. Method

In this section, we describe our approaches to predict queue time.

4.1. Pure Neural Network

In this part, we implement a small experiment to check the feasibility for neural network. We only use a multi-layered linear network and test its functionality simply by comparing with evaluation results from existing models.

4.1.1. Model Structure. This linear model only contains three layer. each size is 2*5, 5*5, 5*1. The model choose request time t_r and request node n_r as input, and use predicted queue time t_p as output. In order to distribute the data evenly, all the data are logarithmic. Detailed formulas are listed below:

$$t_r = int(\log_2(request\_time\_quarter + 1))$$

$$n_r = int(\log_2(request\_node + 1))$$

$$t_a = int(\log_2(actual\_queue\_time\_quarter + 1))$$

t_a is actual queue time quarter, which is used as a label. All time data are represented as quarter, which is chose based on requested time distribution.

4.1.2. Training Details. For training, we use MSE as loss function, and use SGD optimizer. We use theta cluster data as dataset, and use the first 50,000 pieces of data to train it. The loss rate is 10^{-4}, and the batch size is 30. After running 100 epochs, we use the last 20,000 pieces of data to test it.

4.1.3. Evaluation. The loss can directly reflect the results. The loss value is constructed as follow:

$$loss = (log_2(t_a + 1) - log_2(t_p + 1))^2$$

which gives:

$$error\_rate = max\{\frac{t_a}{t_p}, \frac{t_p}{t_a}\} \approx 2^{\sqrt{loss}}$$

error_rate reflects the multiple of the predicted time and the actual time. In the training results, the loss of the test set is 3.36, and the error_rate obtained by calculation is 3.56, which means the predicted time differs the actual time about 2.56 times in average. Actually, it is not too bad for user experience, especially for short-term tasks. For example, for a task predicted by our model to be half an hour, the probability of actual waiting time is 0.14-1.78 hours. It's already possible for users to get the idea of how long the queuing takes.

We compare this model with paper [8] in Figure 8. Light blue represents our results. Although the exact numbers are not comparable (because the data sets are different), our linear model still achieves the same order of magnitude with only two parameters, which indicates that the neural network approach is feasible for this task.

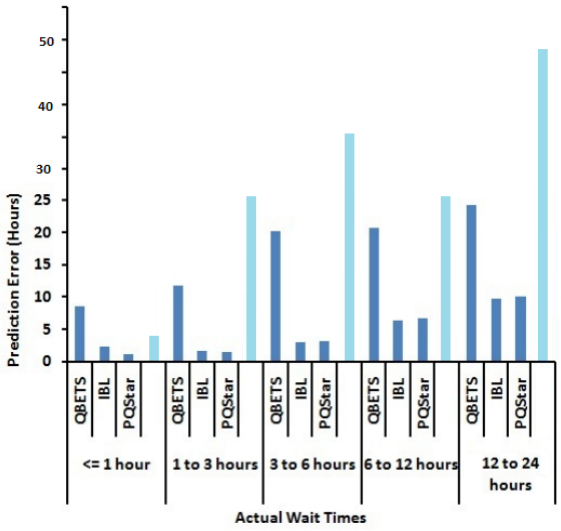

Figure 8. Comparing results with existing models

4.2. Combining Similarity

In the existing work, most of them use similarity job to predict queue time. In recent years, with the development of machine learning, many works have gradually tried to predict queue time by machine learning. Recent studies tend to predict the approximate range of queue time and turn it into a classification problem. Our research goal is also to predict the approximate range

of queue time and improve the accuracy of classification prediction.

The research based on Job Similarity has been proved that queue time can be well predicted. Most works [9] [5] have defined similarity job by calculating Euclidean distance.In recent years, machine learning has also made good progress in Queue Time. A recent work [6] used HMM to achieve about 50% classification prediction accuracy.

Our idea is to combine similarity with machine learning.

Traditional methods for queue time prediction generally follow the IBL [9] architecture and use distance function and induction model. Many works have made some improvements in distance function. Some works use the distance defined by Heterogeneous euclidean-overlap Metric [10] to find similarity jobs, and some use the clustering [11] [12] method to find similarity jobs. In our work, we will use Euclidean distance on higher dimensional space as distance function.

4.2.1. Distance Function. We map the features of a job to the vector space of higher dimension. The similarity between different Jobs can be calculated as follows:

$$Distance = \frac{\sum_{i=1}^{N} \|X_i - Y_i\|}{\sum_{i=1}^{N} \|X_i + Y_i\|}$$

This means that the job has n feature. Each feature has a corresponding value. And we will calculate the Distance between the two jobs according to the above formula. The value of Distance is between 0 and 1. The smaller the Distance is, the higher the similarity is.

4.2.2. Pre-training tuning.. First, we use data set to train machine learning model and save it. It has been mentioned in many articles that the prediction accuracy of machine learning directly on the whole data is low. Because there are a lot of outliers in the data, and some jobs have privileges, which I already mentioned in the data analysis section. Due to the application of similarity job, good prediction results have also been obtained. Therefore, I believe that for a specific job, the information contained in the historical jobs with high similarity is more valuable. Therefore, for a specific job, we want to rely on similarity to weight the data set, and then make the trained machine learning model perform weighted retraining on a part of data with high similarity. The process is shown below Figure 9.

It can improve the accuracy of a particular use case based on the base model, thus improving the overall accuracy. Another problem is that prediction must be time-sensitive. Therefore, the time of retraining and the number of Similarity jobs need to be determined through practice.

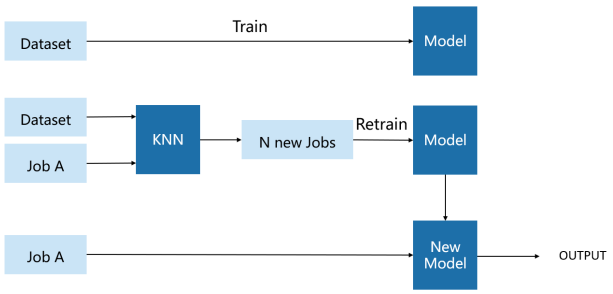

Figure 9. The process of the combine Job similarity and machine learning

4.3. Composite Neural Network

In normal model, the model use all preprocessed data as input, and directly use estimated time as output. There are a few problems in it. First, it is hard to choose useful input. A lot of features that could be processed are useless or too complicated for model to understand. The model need a compressed input, but not too simplified to loss useful information. Second, the time serial need to be considered because the state of the computing clusters could be affected through time. However, merely using RNN would not support its complicated requirements. For example, each task may appear at a random time instead of regular discrete values. If we consider each task as a sample, their interval information will be omitted(or hard to encode), which should be the key information. But if we consider tasks in each single time interval as a sample, then we have to combine multiple tasks information together into one, which means other information in these tasks will have to be omitted.

We denote our proposals from our assumptions for each problems:

Problem 1: Hard to choose input. Instead of using all data together, we focus more on informations that scheduler may get interested in or used for further scheduling. The input must be learnable, say, can be operated properly by an agent to generate output. Since scheduler is a rational agent(we assumed the scheduler's behavior is stable and dependable, or it shouldn't be used at all), the simple approach is to simulate its behavior by neural network. We can set the environment as the input, and the predicted time as output. Notice that if we already know the full strategy of the scheduler, if we do not consider about further enqueuing tasks and decode perfectly for the whole environment, we can get exact correct predicted time. More excitingly, the environment that scheduler accessed is stateless, which means normal NN could solve it.

Problem 2: Time serial information is hard to process. The only difference that made scheduler's behavior deflect is the further enqueuing tasks. Since we already choose the environment as the input, we can simply

encode the "future environment" to represent time serial information. This data can be accessed fully during the training process, so we can easily train the NN mentioned above with completely accurate data. Now we only need a extra model to predict the "future environment".

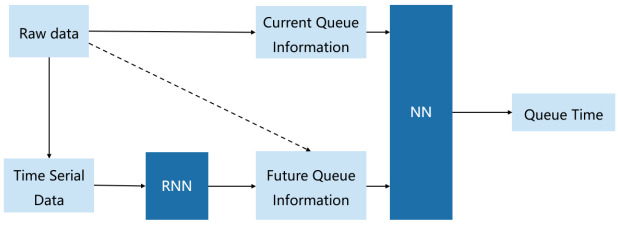

Figure 10. The process of Composite Network

Figure 10. shows running detail of this model. In training process, it obtain current and future queue information to train NN. At the same time, the RNN also would be trained with time serial data. In testing process, these 2 model will work together. RNN first predict future queue information to NN, and NN generate the finial queue time.

5. Future Work

We have talked about some of our experiments and ideas, now, we describe about the goals for this project. For further work, we will consider about following goals:

1) Implement a full model for predicting queuing time in super computers.
2) Comparing with current approaches, try to achieve all overachieve their performance.
3) Validate our model in multiple datasets. If it works, we will try to implant it to SUSTech HPC "Tai-Yi".

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] N. H. Kapadia, J. A. Fortes, and C. E. Brodley, "Predictive application-performance modeling in a computational grid environment," in Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469). IEEE, 1999, pp. 47–54.

[10] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," Journal of artificial intelligence research, vol. 6, pp. 1–34, 1997.

[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] 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.