RESEARCH

Predicting running time of aerodynamic jobs in HPC system by combining supervised and unsupervised learning method

Hao Wang[1], YiQin Dai[2], Jie Yu[3] and Yong Dong[4*]

*Correspondence:
yongdong@nudt.edu.cn
[4]College of Computer, National University of Denfense Technology, Changsha, CHINA
Full list of author information is available at the end of the article

Abstract

Improving resource utilization is an important goal of high-performance computing systems of supercomputing centers. In order to meet this goal, the job scheduler of high-performance computing systems often use backfilling scheduling to fill short-time jobs into the gaps of jobs at the front of the queue. Backfilling scheduling needs to obtain the running time of the job. In the past, the job running times are usually given by users and often far exceeded the actual running time of the job, which leads to inaccurate backfilling and a waste of computing resources. In particular, when the predicted job running time is lower than the actual time, the damage caused to the utilization of the system's computing resources becomes more serious. Therefore, the prediction accuracy of the job running time is crucial to the utilization of system resources. The use of machine learning methods can make more accurate predictions of the job running time. Aiming at the parallel application of aerodynamics, we propose a job running time prediction framework SU combining supervised and unsupervised learning, and verifies it on the real historical data of the high-performance computing systems of China Aerodynamics Research and Development Center(CARDC). The experimental results show that SU has a high prediction accuracy(80.46%) and a low underestimation rate(24.85%).

Keywords: High performance computing; Job scheduling; job running time prediction; Machine learning; Prediction accuracy; underestimation rate

Introduction

High-performance computing [1] has been widely used in the fields of science and engineering. The explosive growth of scientific computing demand [2] provides a good growth environment for the rapid development of high-performance computing. At the same time, it also puts forward higher requirements for high-performance computing systems. High-performance computing has the characteristics of powerful computing capability and high scalability. How to ensure or even improve the preset service level agreement (SLA) [3] and computing resources more rationally are also very important issues.

On high-performance computing systems, the job scheduler is a very important part. It is responsible for scheduling the jobs submitted by users according to a certain strategy. To improve the utilization of computing resources, the job scheduler usually adopts a backfilling strategy, which scheduling short-time jobs at the back of the job queue in advance if these jobs don't delay the execution of the first job in the queue. Usually in this process, the job scheduler capture the following job

information: the submission time of the job (Submit_time), the number of CPU cores required for the job to complete (CPU_req), the job name (Job_name), the user name (User), the user ID (User_id), the job waiting time (Wait_time) and estimated time for the job (Time_req), etc. Job backfilling scheduling depends on the estimated time of the job. In the past, the estimation of the running time of the job was provided by users, but the work of Cirne et al. [4] showed that the estimated time given by users of more than half of the job is five times or more than the actual running time of the job. This behavior is in line with the user's desire to complete the job normally because a running job will be killed if the time allocated to it has been exhausted. Moreover, many users may not have sufficient knowledge about the job and the operating environment and tend to give the job a running time that far exceeds normal requirements. But for the system, this approach is not beneficial and can cause a lot of waste of computing resources.

Using machine learning methods to predict the running time of the job can effectively improve the inaccuracy of the time provided by users. Wu [5] et al. proposed the IRPA algorithm, which greatly improved the accuracy of the running time prediction of a single job. Pumma [6] et al. proposed a meta-heuristic prediction algorithm based on a dynamic environment, which also achieved good results. Xiao [7] et al. proposed the GA-Sim algorithm based on the idea of clustering, which improved the prediction accuracy and underestimation of operating time.

Aerodynamics applications are typical applications in high performance computing, mainly used to study the force analysis and possible physical and chemical changes when aircrafts and other equipment move relative to air or other gases. With the rapid development of more than half a century, this subject has become increasingly mature and serves many fields such as aviation, aerospace, shipbuilding, and water conservancy. Aiming at the parallel applications of aerodynamics, predicting their running time is of great significance for improving the throughput of the aerodynamic HPC system and improving the efficiency of the system.

For the parallel application of aerodynamics, we comprehensively consider its computing characteristics and proposes a job running time prediction framework SU (Supervised and Unsupervised learning method) combining supervised learning and unsupervised learning. In SU, supervised learning refers to a machine learning method with regression as the target value. We tried three supervised learning methods in SU. Unsupervised learning refers to clustering, which is a machine learning method without a target value. SU is implemented in two steps. In the first step, the jobs in the history log are clustered to obtain multiple clusters of similar jobs $S_1^{'}$, $S_2^{'}$, ... Then, for the job to be predicted, find its k nearest neighbor jobs in its cluster (Suppose it is $S_1^{'}$), and get a set of similar jobs $S_1^{''}$. The second step adopts a supervised learning method to predict the running time of the job. The results of verification on the real historical data of the high-performance computing systems of CARDC show that the prediction accuracy reaches 80.46%, and the underestimation rate is 24.85%.

The main contributions of this paper include the following three aspects:

- Firstly, we propose the SU, a job time prediction framework, which combines the advantages of supervised learning methods and unsupervised learning methods and lays a foundation for the job running time prediction;

- Secondly, we integrate three machine learning regression models (linear regression, random forest regression and support vector regression) into the SU framework to realize the prediction of job running time;
- Thirdly, based on the real historical data of the high-performance computing systems of CARDC, the above algorithm is verified and tested. The results show that when the SU framework adopts the SVR method, it has good accuracy and underestimation rate.

The main structure of this paper is as follows: The second section introduces the scheduling and backfilling algorithms and several algorithms for predicting the job running time, and analyze their characteristics. The third section describes the job running time prediction framework SU proposed in this paper in detail. In the fourth section, we run the SU framework on the production log provided by CARDC for verification. Finally, we summarize our work and future challenges.

Related Work

Scheduling algorithms usually include FCFS (first come first served), SJF (short job first), LJF (long job first), and WFP [8]. The common practice is to use FCFS to schedule jobs. As the name suggests, this strategy sorts the scheduling order of jobs according to their arrival order. FCFS is simple, fair, and easy to implement, but when idle resources cannot meet the needs of jobs at the top of the queue, resources can be wasted. In addition, short jobs may need to wait until long jobs are completed, resulting in a longer average waiting time, so this method makes resource utilization very low.

The SJF scheduling algorithm sorts the jobs in the increasing order of the estimated running time and then delivers them to the system for scheduling. this scheduling algorithm is beneficial to the situation where there are many short jobs because a large number of short jobs in the queue can be scheduled in advance. In extreme cases, long jobs in the queue may need to wait forever. On the contrary, for the LJF scheduling algorithm, its advantages and disadvantages are exactly the opposite to those of SJF. For the WFP scheduling algorithm, it schedules the jobs in descending order of the score after scoring the jobs according to a certain formula 1:

$$Score = CPU\_req * (Wait\_time/Time\_req)^3 \tag{1}$$

The WFP scheduling algorithm comprehensively considers the demand for computing resources of the job and the waiting time of the job in the job queue. As the waiting time for jobs increases, the job becomes more thirsty and the score grows, which means it may executed in advance at some point. This is a very interesting improvement. In the previous scheduling algorithm, the problem of job thirst was not considered.

In the scheduling system, in addition to the job scheduling algorithm, there is usually backfilling of jobs, that is, some small jobs at the back of the job queue are executed in advance according to a certain algorithm without affecting the execution of the jobs at the front of the queue. Backfilling can further utilize idle computing resource fragments, thereby improving resource utilization. There are mainly conservative backfilling and EASY backfilling [8]. For conservative backfilling, as long

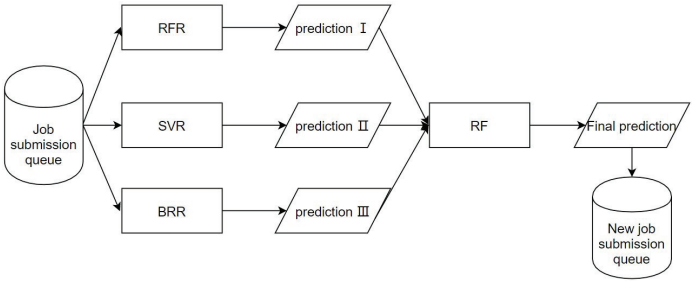

Figure 1 IRPA

as the small backfilled job does not delay the previous jobs in the waiting queue, the backfilling will be executed. Compared to conservative backfilling, the EASY method is more radical. As long as the scheduled small job does not delay the first job in the queue, it will be scheduled. The two types of backfilling have their own merits. In the former, there is no job thirst. In the latter, jobs may experience unbounded delays. Usually in HPC, similar jobs will be called repeatedly, so you can choose a suitable backfilling method based on the actual job scheduling situation.

Considering the importance of the accuracy of the estimated job running time in the high-performance computing systems for job scheduling, more and more work is devoted to improving the accuracy of the job running time and reducing the underestimation rate on this basis. The main effect of reducing the underestimation rate is to reduce the serious waste of resources caused by the underestimation of the system.

In the study of job running time prediction, Tsafrir et al. [9] first proposed the Last-2 method. This method is very simple. It takes the average of the actual running time of the last two submitted jobs by the same user as the predicted time of the newly job submitted by this user. However, the accuracy and underestimation rate of this method cannot meet expectations.

Wu [5] et al. proposed an integrated learning model IRPA based on specific types of jobs (VASP). The model is shown in Figure 1. This model combines a variety of machine learning methods and takes into account some unique characteristics of the VASP. In this model, firstly uses three sub-models including Random Forest Regression, Support Vector Regression, and Bayesian Ridge Regression to predict the job time respectively. Then, the three predicted values obtained are used as the input of the random forest classification model. Finally, the prediction result was determined by these four machine learning methods. The IRPA model has achieved high accuracy for VASP, but it does not take into account the underestimation of the predicting time.

Pumma et al. [6] proposed a meta-heuristic optimization algorithm with classification and linear regression technology based on a dynamic environment to predict job running time. The method is mainly divided into three steps: firstly, when the job arrives, it uses MICA to extract some job features related to running time and uses perf to create a workload profile (including branch prediction, multi-instruction, etc.). Then, the result obtained in the first step is fed back to the decision tree to classify the job. In the experiment, there are 7 classes in total. Finally, the regression model of the corresponding category is used to predict the operating time.

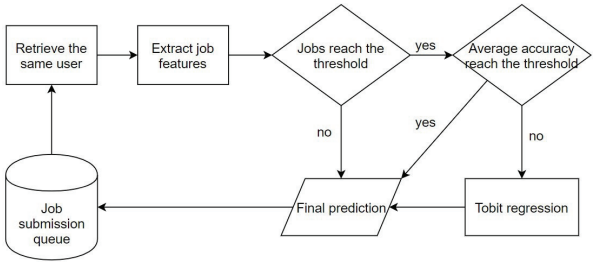

Figure 2 TRIP

In the last step, the author uses the ABC algorithm (artificial bee colony algorithm) to find the relationship between input attributes and running time, which is a meta-heuristic algorithm. The model has achieved good results, but there are some limitations to this method. First, the job is divided into seven categories, but once the newly arrived job does not belong to one of the categories, the method will fail. Second, it is a method based on a dynamic environment and needs to consider whether the user and the system can accept the time spent in the predicting process.

In [10], Fan Y et al. mainly started from the point of view of data processing, hoping to optimize and filter data to improve the prediction accuracy of job running time. A lot of work before this has been devoted to improving the prediction accuracy by reducing the overestimation of running time but did not solve the problem of underestimation. As we know, underestimating time means disastrous consequences. Unfortunately, these two goals are in conflict, and improving the prediction accuracy of overestimated jobs may increase the probability of underestimation. Fan Y et al. proposed an online adjustment framework TRIP starting from data. TRIP uses the data truncation capability of the Tobit model. In TRIP, data that does not meet the conditions are truncated, so that more effective data brings more accurate running time predictions. The TRIP model is shown in Figure 2. For each incoming job, TRIP first searches the library for historical information with the same user, project, and job. Then, if the number of information found is less than the threshold, the time provided by the user will be used. Otherwise, it compares whether the average accuracy of historical jobs is greater than a certain threshold. If it is, the time prediction provided by the user will be used, otherwise, the Tobit model will be used for prediction. Experimental results prove that this method significantly reduces the underestimation rate but the average accuracy does not change much.

In [11], Guo J et al. considered the problem of data categories and believed that considering the overall performance index is unfair for the data with a small number. Therefore, an evaluation standard for a few types of data is proposed. In the experiment, Guo J et al. used XGBoost and RF machine learning methods to classify and predict data and then proposed evaluation criteria for a small number of data. At the same time, it was also found that when the jobs were classified by application name, higher recall rates, precision rates, and F1 scores were obtained. However, this method fails when the user is not required to provide an estimated time for the job.

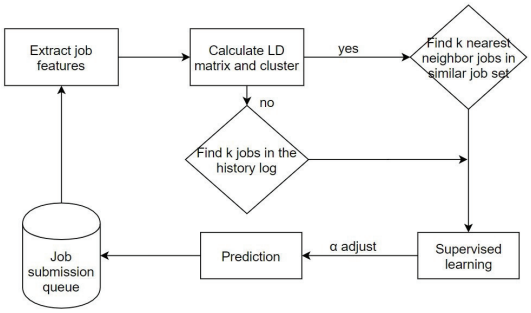

Figure 3 SU Predicting framework.

Table 1 Job features

Features	Meanings
Elapsed	The actual running time of the job
CPU_req	The number of CPU cores required
Submit_time	Job submission time
Job_name	Job name
User	User name

It can be found that when using machine learning methods to predict the running time of a job, many aspects need to be taken into account at the same time. In response to the problems in related work, we designed the SU framework, taking into account the prediction accuracy and underestimation issues. At the same time, the generalization performance of the algorithm is better.

SU Predicting framework

The SU framework is implemented based on a combination of supervised learning and unsupervised learning. Supervised learning methods mainly include classification and regression. Our numerical prediction of job time is a regression method. In order to obtain high-precision prediction results, we must first extract effective data and features, which is difficult to achieve by supervised learning alone. Therefore, we combine clustering, a typical unsupervised learning method, cluster similar things together by calculating similarity and then provide effective data dependence for supervised learning. As shown in Figure 3, SU is mainly composed of clustering, k-nearest neighbors, and prediction.

- Through clustering of unsupervised learning, similar jobs in the history log are clustered together, and multiple similar job sets $S_1^{'}$, $S_2^{'}$,... are obtained. Here we use the shortest edit distance algorithm on User and Job_name to calculate the distance matrix and use this as the standard of clustering to obtain the set of similar jobs $S^{'}$.
- According to the job to be predicted, SU selects its k nearest neighbor jobs in the corresponding set $S^{'}$ to form similar job set $S^{''}$, and uses it as a training set.
- SU uses supervised learning to estimate the running time of the job. It should be noted that when a new user appears, that is, the job to be predicted does not belong to any previous cluster. Then we look for its k neighbor jobs in the historical log and predict running time with these jobs. The job features used in SU are shown in Table 1.

Table 2

		s	o	n
	0	1	2	3
s	1			
u	2			
n	3			

Table 3

		s	o	n
	0	1	2	3
s	1	0		
u	2			
n	3			

Table 4

		s	o	n
	0	1	2	3
s	1	0	1	2
u	2	1	1	2
n	3	2	2	1

The shortest edit distance matrix LD

The shortest edit distance algorithm is the first step to realize the SU framework. As shown in figure 3, we calculate the LD matrix for clustering. It is the basis needed to completing the clustering, and it can better measure the distance between strings. Here we use the Levenshtein distance [12] to define the edit distance, which can be used to indicate how many steps string A needs to undergo to transform into string B. There are three editing operations: add, delete and replace. To transform a string into another, three operations are generally used in combination. What we want to get is the shortest edit distance from string A to string B, which describes the degree of similarity between the two strings. The shorter the edit distance between two strings, the higher the similarity between them.

We use dynamic programming to measure the shortest edit distance between two strings. In SU, we combine User and Job_name to form a new string User_job. We select two samples from this new feature. Since the samples are very long, they are cut into the string 'son' and 'sun'. The process of solving the shortest edit distance is as follows:

Initialize the matrix of string A and B(shown in Table 2).

Calculate the value of position (1,1)(shown in Table 3): at this time 's'='s', so the value of (1,1) is min1+1,1+1,0+0=0, that is, the left number of the position (1,1) plus 1, the upper number plus 1, and the upper left number plus 0, then take the minimum value. It should be noted that when the compared two characters are not the same, the numbers in the three positions are all increased by 1.

Repeat the rule from the upper left corner to the lower right corner to calculate the value of the remaining positions until the matrix is filled(shown in Table 4).

Finally, the value 1 in the lower right corner is obtained, which is the shortest edit distance between string A and string B. This is consistent with the phenomenon that only needs to replace an intermediate character to complete the transformation.

We refer to the above dynamic programming method to calculate the shortest edit distance between each sample and obtain an n*n LD distance matrix. Here, n is the number of jobs, and LD is a symmetric matrix whose main diagonal is all 0. In actual calculations, only the upper-half angle or the lower half-angle need to be calculated. Such a matrix LD is the sample input when the job is clustered.

Using k-means++ clustering to obtain similar job set S'

After obtaining the LD matrix, we use the matrix to clustering. Cluster analysis is a method of unsupervised learning, which aims to discover the relationship between them in the data, and group these data. The greater the similarity of data within a cluster and the smaller the similarity of data between clusters, the better the clustering effect. This is not the same as classification. Clustering gathers similar data together and gives each cluster of data a label. It does not care about this label when dividing, and even the number of labels is uncertain. The classification is when the number of labels has been determined, and it is hoped that a classifier will be used to classify the data into suitable labels. This is a method of supervised learning. Data clustering mainly includes Partition-based Methods, Density-based methods, and Hierarchical Methods. Here we use the classic Partition-based clustering methods, k-means++ [13].

k-means++ is evolved on the basis of k-means [14]. The principle of the k-means algorithm is relatively simple and the convergence speed is fast. Its time complexity is close to linear, so in the case of relatively large sample size in the SU framework, the efficiency is relatively high. But the k-means algorithm also has certain defects. First, the number of clusters k needs to be given in advance, which requires certain prior knowledge. The second is that in k-means, it is necessary to set a clustering center to perform the initial division at the beginning, and then optimize the division, so the choice of the initial focus will have a greater impact on the clustering results, which may cause damage to the effect of clustering.

The random allocation of initial centroids in k-means will lead to the problem that the final result may only be a local optimal value. In the SU framework, the k-means++ algorithm is used to cluster the sample points. Its basic idea is that the initial clustering centers should be sufficiently far apart, and usually, multiple clustering is performed, each time a different center is initialized, and the smallest inertial clustering result is used as the final clustering result. The calculation formula of inertial is as follows:

$$inertial = \sum_{i=1}^{k} \sum_{(x \in E_i)} ||x - u_i||_2^2, \tag{2}$$

Among them, u_i is the center of each cluster, x represents the sample point. Inertial is a measure of the degree of aggregation within a cluster. The smaller the value, the higher the similarity of samples in the cluster.

In the SU framework, we calculate the LD matrix on the data obtained from CARDC and then use k-means++ for clustering to obtain similar job sets. Considering that the amount of data is relatively large and each cluster in the final clustering result belongs to the same user, the sample points of each user are clustered separately, which greatly reduces the clustering time. When clustering, the interval of k can be specified according to certain prior knowledge, and the best k value can be calculated. Usually, we use SSE and Calinski-Harabasz index to measure the pros and cons of the k value. SSE is the within-cluster sum of squared errors, which can be calculated by Equation 2. The smaller the value is, the better

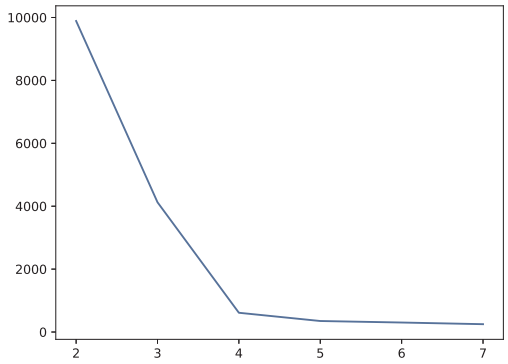

Figure 4 SSE changes with the value of k.

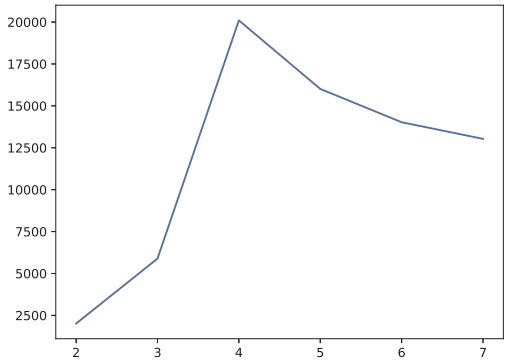

Figure 5 Calinski-Harabasz changes with the value of k.

the k is. Calinski-Harabasz considers the density of clusters and the degree of dispersion between clusters. Its calculation process is also very simple. The larger the score of Calinski-Harabasz is, the better the k is. The formula is as follows:

$$C_{score} = tr(B_k)/tr(W_k) * (n-k)/(k-1), \tag{3}$$

Among them, m is the number of samples, n is the number of clusters, B_k is the covariance matrix between cluster samples, W_k is the covariance matrix of the samples in the cluster, and tr is the trace of the matrix. A higher Calinski-Harabasz score means that the covariance of the samples in the cluster is smaller and the covariance of the samples between clusters is larger. Both of them indicate that the clustering effect is better.

Figures 4 and 5 are the clustering results of a sample of a certain user. The definition of SSE measures the similarity of the samples in the cluster. The lower the value, the higher the sample similarity. The k value when the elbow appears in the figure is generally a good choice. For Calinski-Harabasz, it measures the cluster density and dispersion between clusters of the sample. Generally, the value of k corresponding to the high Calinski-Harabasz value is selected. It can be seen from the figure that SSE has an elbow shape when k=4 and Calinski-Harabasz take the maximum value at this time. Therefore, for this user's sample, when k=4, the

clustering effect is the best. In the realization of SU, we finally group the samples into 15 clusters according to the shortest edit distance matrix LD.

Using k-nearest neighbors to obtain similar job set $S^{''}$

After k-means++ is used to obtain the set of similar jobs $S^{'}$, the idea of k-nearest neighbor is used to obtain the k nearest neighbors of the job to be predicted. We select two numerical attributes of the jobs to calculate the similarity between them. The two attributes are the number of CPU cores required by the job CPU_req and the time the user submits the job Submit_time. Among them, CPU_req is directly related to the job running time. The use of Submit_time is to hope that the machine learning method can learn some of the users' submission habits. For example, the user may like to submit some larger jobs at night, and submit smaller jobs during the day. Of course, the Submit_time feature needs to be processed. Firstly, it needs to be modulated from seconds to hours. Secondly, considering that it is a cyclic variable and its size is relatively meaningless, it should be coded.

For the coding of cyclic characteristics, the angle in the polar coordinate system is used to describe each value of cyclic characteristics. At the same time, the angle can be uniquely determined by using the sin function and cos function, then we get Submit_sin and Submit_cos. Finally, after standardizing CPU_req, Submit_sin, and Submit_cos, they are denoted as (c, ss, sc), and formula 4 is used to calculate the similarity S_{score} between job i and job j.

$$S_{score} = \sqrt{(|c_i - c_j|^2 + |ss_i - ss_j|^2 + |sc_i - sc_j|^2}, \tag{4}$$

After calculating the job similarity, select k jobs with the smallest S_{score} value. In this way, after the two steps of clustering and k-nearest neighbors, the job similarity set $S^{''}$ of the job to be predicted is formed.

Using supervised learning to predict job time

After obtaining the similar job set $S^{''}$, we use k jobs that contain CPU_req, Submit_sin, Submit_cos, and the actual job running time Elapsed to train the prediction model of the job to be predicted. Among them, the first three features are used as model inputs and Elapsed is used as model output. In our experiment, we tried different regression methods in SU's job time prediction period and chose the best method as the prediction model.

In the final prediction time expression, an adjustment factor is introduced, and the value is determined by the best experimental effect. It can greatly reduce the underestimation rate of the prediction under the condition of losing a little prediction accuracy. We know that the prediction accuracy describes how close the predicted job time is to the actual running time, and the underestimation rate refers to the proportion of the job whose predicted time is lower than the actual time. When the prediction time interval is multiplied by a coefficient greater than 1, the entire value interval will be larger. However, relative to the loss of prediction accuracy, the underestimation rate has been reduced more, indicating that a considerable part of the prediction results is located below the actual running time but very close to it. Thus, the final result can be expressed as:

$$Final = \alpha * Predict(job\_new) \tag{5}$$

Experiment

In the experiment, we implemented the SU framework and used real logs obtained from CARDC to compare the effects of the three machine learning methods of linear regression, random forest regression, and SVR with SU framework and without SU framework. Experiments have shown that when the SVR method is used in the prediction period of the SU framework, it has the best prediction accuracy and underestimate rate.

Experimental platform and data set

Several machine learning methods used in the experiment and the standardization of data processing are all provided by the scikit-learn library [15].

The data required for the experiment is provided by CARDC, and contains a total of 30,481 original samples generated in three months.

Before using SU to make predictions, we preprocessed the data. Firstly, we remove the unsuccessful jobs, that is, the job did not finish within the specified time, and then we remove the jobs missing the features shown in Table 1, leaving 26,208 valid jobs.

Model evaluation criteria

In this article, three evaluation indicators are used, namely Average Predictive Accuracy(APA), Mean Absolute Error (MAE) and Underestimate Rate (UR).

APA describes the average prediction accuracy of all jobs to be predicted, that is, how close the overall prediction is to the actual job time. The calculation formula for a single job to be predicted is as follows:

$$APA_i = \begin{cases} Pre_i/Elapse_i, & Pre_i < Elapse_i \\ Elapse_i/Pre_i, & Pre_i \geq Elapse_i \end{cases} \tag{6}$$

Among them, Pre_i represents the predicted job running time of job i, and $Elapsed_i$ represents the actual running time of the job. In this way, the average prediction accuracy APA of all jobs can be expressed as:

$$APA = (\sum_{i=1}^{n} APA_i)/n \tag{7}$$

Among them, n is the total number of jobs to be predicted, and the value of APA is between 0 and 1. The closer to 1, the better the prediction effect.

MAE describes the difference between the predicted time of all jobs and their true values, and its expression is:

$$MAE = (\sum_{i=1}^{n} |Pre_i - Elapsed_i|)/n \tag{8}$$

MAE and APA are complementary to each other.

UR describes the proportion of jobs that are predicted to be lower than the actual job time. Such a prediction is meaningless and has greater damage to the

Table 5 Predicting results of six methods.

Method	Linear	RFR	SVR	Single-Liner	Single-RFR	Single-SVR
MAE	3140.5	3181.9	2762.7	15039.3	10892.6	9488.0
APA	0.7482	0.7469	0.8046	0.2784	0.3769	0.5400
UR	0.3134	0.3022	0.2485	0.2871	0.2731	0.489

utilization of system computing resources. It should be reduced as much as possible. The calculation formula is as follows:

$$UR = (\sum_{i=1}^{n} I(Pre_i < Elapsed_i))/n \tag{9}$$

Where I is an indicative function, defined as follows:

$$I(Pre_i < Elapsed_i) = \begin{cases} 1, & Pre_i < Elapse_i \\ 0, & Pre_i \geq Elapse_i \end{cases} \tag{10}$$

Experimental steps

- Filter the data according to the requirements of the data processing period.
- After calculating the shortest edit distance matrix, cluster the samples.
- Divide the jobs, get the training set and the test set, and get its neighbor job set according to the job in the test set.
- Predict the job to be predicted.
- Evaluate the results of the experiment.

Experimental results and analysis

In the job time prediction period, we compared the effects of three machine learning methods: linear regression, random forest regression, and SVR with SU framework and without SU framework. When the SU framework is not used, we do not perform clustering and k-nearest neighbor operations on samples. After extracting CPU_req, Submit_time, and Elapsed, we perform cycling encoding on Submit_time. Then we standardize CPU_req, Submit_sin, and Submit_cos, and finally, predict the running time of the job.

The values of the mean absolute error MAE, average prediction accuracy APA, and underestimation rate UR obtained in the experiment are shown in Table 5. The Method column represents the machine learning method used in the prediction period, and the three on the left are linear regression, random forest regression, and support vector regression with SU framework, and the three on the right are linear regression, random forest regression, and support vector regression without SU framework.

Figure 6 shows the comparison of the mean absolute error under six different conditions. When the SU framework is used, the average MAE obtained by the three machine learning methods is 3028, which is 74% lower than the average without the SU framework. This indicates that when the SU framework is used, the predicted value has higher stability, and the MAE value is the lowest when using the SVR prediction method with the SU framework.

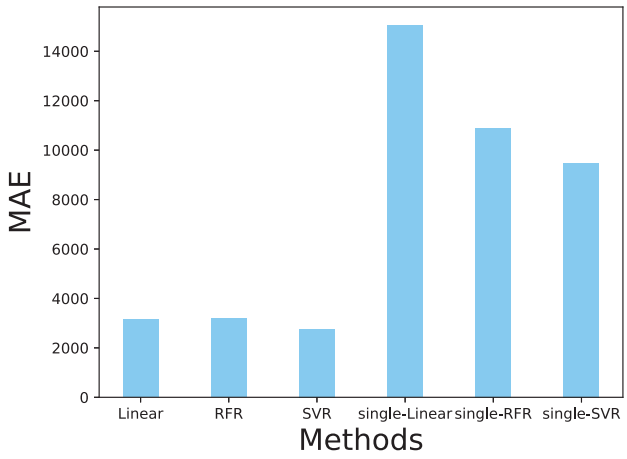

Figure 6 MAE comparison.

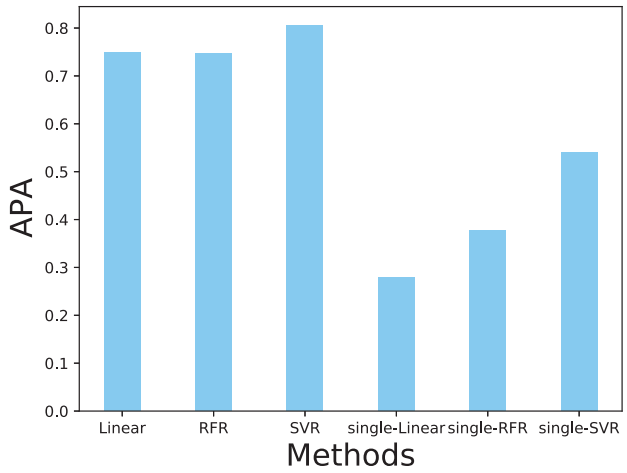

Figure 7 APA comparison.

Figure 7 shows the comparison of prediction accuracy under six different conditions. When the SU framework is used, the average APA obtained by the three machine learning methods is 76.66%, which is 36.8% higher than when the SU framework is not used. This means that when the SU framework is used, the prediction have higher accuracy, and the APA value is the highest when using the SVR prediction method with the SU framework.

Figure 8 shows the comparison of the underestimation rate under six different conditions. The underestimation of the first five methods is relatively close, but the SVR method with SU framework still obtains the lowest underestimation rate.

The SVR algorithm has the characteristics of good generalization performance and fast convergence speed. On the three indicators, SVR with framework has the best performance. This shows that the combination of the SU framework and SVR not only reduces underestimation but also ensures the accuracy of the prediction.

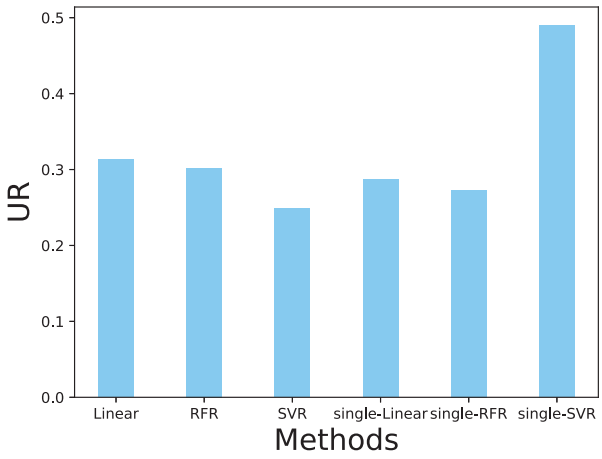

Figure 8 UR comparison.

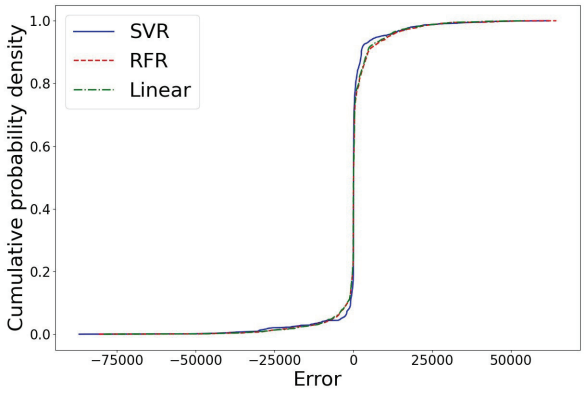

Figure 9 Cumulative error distribution.

On the basis of the previous work, we analyzed the cumulative distribution of the error of predicting the job running time with the framework, as shown in Figure 9.

The left and right sides of the vertical line x=0 represent the underestimation and overestimation of the job running time prediction. It can be seen that on the left side of x=0, RFR and Linear are relatively close, but the underestimation of SVR is much better. At the same time, when approaching the vertical line of x=0, the change of SVR is more significant, indicating that there are many underestimations that are close to the actual job running time. Therefore, it is necessary to adjust the final prediction results with . Compared with the loss of prediction accuracy, the reduction of underestimation is more obvious. On the right side of x=0, SVR is also closer to the real job running time than the other two algorithms.

Furthermore, we analyze the prediction results when using the SVR method under the SU framework according to the job classifications. The prediction results of 15 job classifications are shown in Figures 10. For I, K, and L, these three classifications of jobs have the best comprehensive performance on the three evaluation criteria. Among them, taking job L as an example, its CPU requirements are 24, 64, and 96 respectively. At this time, the corresponding maximum-to-minimum ratios of real

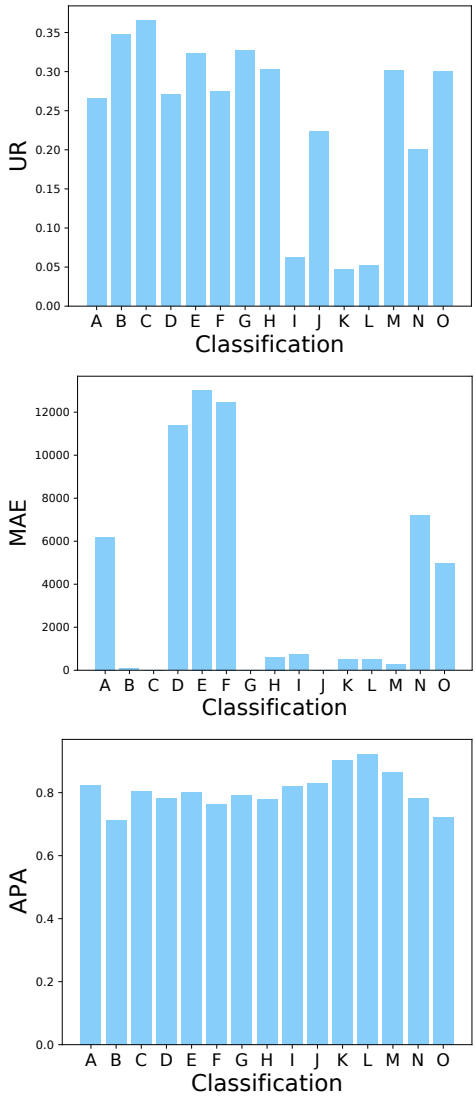

Figure 10 Results with job classifications

job running time are 1.40, 1.01, and 1.37 respectively. The poor overall performance of job B has a CPU demand of 16, and the maximum-to-minimum ratio of the real job running time is 12.7. It can be found that in the case of a fixed job classification and a fixed CPU demand, the fluctuation of the real job time has a great influence on the prediction result. The smaller the fluctuation range, the higher the prediction accuracy and the smaller the underestimation. The larger the fluctuation range, the lower the accuracy and the higher the underestimation. The same pattern can be found in the remaining classifications of homework.

Finally, we performed simulated backfilling scheduling experiments on the above six prediction results on the Slurm simulator [16] to verify its scheduling performance. The Slurm simulator supports EASY backfilling and FCFS scheduling algorithms. We use Average Waiting Time(AWT) to evaluate scheduling performance under these two strategies. The AWT under the six methods are shown in Table 6,

Table 6 Backfilling perfermance.

Method	Linear	RFR	SVR	Single-Liner	Single-RFR	Single-SVR
AWT	213.8	204.6	192.6	297.4	285.1	274.6

from left to right are linear regression, random forest regression, and support vector regression under the SU framework and linear regression, random forest regression, and support vector regression under the non-SU framework. It can be found that the AWT of the three methods when the SU framework is used is 29% higher than that of the three methods when the SU framework is not used. Among them, when the SVR prediction method is used under the SU framework, the scheduling performance is the best.

In summary, the SU framework can guarantee higher prediction accuracy and lower underestimation rate without requiring users to provide estimated job running time. In the high-performance computing systems of the Aerodynamics Research and Development Center, most of the operations are CFD-related and there are relatively few users. At the same time, applications are relatively concentrated. In this scenario, it is also more conducive to the play of the SU framework.

Conclusion

Although the SU framework has achieved good results in the real history log of aerodynamics, the author believes that SU has room for improvement in prediction accuracy, average absolute error and underestimation. Because some of the key job parameters are in the configuration file when the user submits the job, such as the number of grids. The number of grids represents the division unit for force analysis. If the number of grids is larger, more calculations are required, so it has a close relationship with the job running time, but this type of data was not obtained in this experiment. When the job classification and CPU demand are fixed, there is a close relationship between the real job running time fluctuations and these parameters. So in order to improve the prediction effect, it is hoped that this type of character data can be captured in the future, which is believed to bring better prediction results and further enhance the utilization of computing resources.

Declaration

Acknowledge

I would like to thank Mr. Dong Yong and Mr. Yu Jie for their support and help in topic selection, data collection, experimental design, and the completion of the paper. This work was carried out at the Computing Center of CARDC.

Authors' contributions

DY and YJ put forward algorithm ideas and provide guidance and an experimental environment. WH was responsible for improving the SU framework, conducting experiments, and completing the first draft, DY and DYQ were responsible for revising and finalizing the draft.

Funding

This paper is supported by National Numerical Windtunnel project, project number 2018-ZT6B13.

Availability of data and materials

The data supporting the results of this study can be obtained from the Computing Center of CARDC, but the availability of these data is limited, and these data have been used with permission.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]College of Computer, National University of Denfense Technology, Changsha, CHINA. [2]College of Computer, National University of Denfense Technology, Changsha, CHINA. [3], China Aerodynamics Research and Development Center, Mianyang, CHINA. [4]College of Computer, National University of Denfense Technology, Changsha, CHINA.

References

1. Wang, R., Lu, K., Chen, J., Zhang, W., Li, J., Yuan, Y., Lu, P., Huang, L., Li, S., Fan, X.: Brief introduction of tianhe exascale prototype system. Tsinghua Science and Technology **26**, 361–269 (2021). doi:10.26599/TST.2020.9010009
2. Liao, X.: Milkyway-2: back to the world top 1. Frontiers Comput. Sci. **8**(3), 343–344 (2014). doi:10.1007/s11704-014-4901-0
3. Leff, A., Rayfield, J.T., Dias, D.M.: Service-level agreements and commercial grids. IEEE Internet Comput. **7**(4), 44–50 (2003). doi:10.1109/MIC.2003.1215659
4. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. In: IEEE International Workshop on Workload Characterization (2001)
5. Gui-Bao, W.U., Shen, Y., Zhang, W.S., Liao, S.S., Wang, Q.Q., Jing, L.I.: Runtime prediction of jobs for backfilling optimization. Proc of the 13th HPC China, 403–408 (2017)
6. Pumma, S., Feng, W., Phunchongharn, P., Chapeland, S., Achalakul, T.: A runtime estimation framework for ALICE. Future Gener. Comput. Syst. **72**, 65–77 (2017). doi:10.1016/j.future.2017.02.040
7. Xiao, Y.H., Lun-Fan, X.U., Xiong, M.: Ga-sim: A job running time prediction algorithm based on categorization and instance learning. Computer Engineering & Science **6**, 987–992 (2019)
8. Weil, A.M., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distributed Syst. **12**(6), 529–543 (2001). doi:10.1109/71.932708
9. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel Distributed Systems **18**(6), 789–803 (2007)
10. Fan, Y., Rich, P., Allcock, W.E., Papka, M.E., Lan, Z.: Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER) (2017)
11. Machine Learning Predictions for Underestimation of Job Runtime on HPC System. Department of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan (2018)
12. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Dokl. Akad. Nauk SSSR, 1965 **10**(8), 707–710 (1966)
13. SARUNYA, K.: An extended k-means++ with mixed attributes. Latest Advances in Information Science and Applications, 131–135 (2012)
14. A.M, F., A.M, S., F.A, T., M.A, R.: An efficient enhanced k-means clustering algorithm. A **7**(10), 1626–1633 (2006)
15. Scikit-learn library[EB/OL] (2018-12-11). www. scikit-learn. org
16. Simakov, N.A., Innus, M., Jones, M., Deleon, R.L., White, J.P., Gallo, S.M., Patra, A.K., Furlani, T.R.: A slurm simulator: Implementation and parametric analysis. In: International Workshop on Performance Modeling (2017)