

OKCM: Improving Job Scheduling of High-Performance Computing using Efficiency Runtime Prediction Model^{*}

Jingbo Li, Xingjun Zhang^{*}, SomeOne1 and SomeOne2

School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

ARTICLE INFO

Keywords:

High performance computing
Parallel job scheduling
Runtime prediction
Online learning
KNN
Correction mechanism

ABSTRACT

Job scheduling is becoming increasingly important in large-scale high-performance computing (HPC) systems as the parallel scale, number and types of jobs continue to increase. Job-priority policies and backfilling mechanisms are the most useful practice for improving scheduling performance. Especially, job runtime is an important element in these methods, and precise knowledge of this parameter can significantly improve the system performance. Previous studies and models have focused only on improving accuracy, resulting in higher computational and time overhead and difficulties in real-time scheduling system deployment. Thus, an efficient runtime prediction model, referred to as online learning and K-nearest neighbors based (KNN-based) predictor with correction mechanism (OKCM), is proposed. OKCM updates in real time through online learning and is friendly to new users and users with a small data accumulation, owing to the KNN-based predictor. What's more, OKCM guarantees safe execution of jobs through the correction mechanism. To evaluate our model, we designed and implemented a trace-driven simulator, HPCsim, based on real job workloads. The results demonstrated that OKCM can achieve high accuracy, up to 82.3%, with a low overhead. Furthermore, OKCM can achieve significant scheduling performance improvements and can be used to enhance primary prioritizing and backfilling methods without scheduling limitations.

1. Introduction


Job scheduling system is a critical middleware for high-performance computing (HPC) platforms which is responsible for managing and scheduling jobs [1]. An excellent scheduling system can effectively improve the performance of the HPC platform and reduce the average user waiting time, ensuring fairness between large and small jobs without starvation and the provision of high-quality services to users. Users submit their jobs by a command to the centralized waiting queue managed by the job scheduler. A submit command contains all the information necessary to run the job, including requested nodes, requested job runtime, and job name. The job scheduler periodically checks job queues and HPC resources, and determines the job order in the queues [2]. The scheduling system decides when and where to execute the jobs. Job-priority policies order jobs based on job attributes such as job arrival time, job runtime estimate, and job size. For example, the shortest job first policy (SJF) orders jobs by job runtime in increasing order, which implies that the shorter a job is, the higher its priority will be. In many policies [3, 4], the runtime is a crucial factor for scheduling [5].

To increase the utilization of HPC resources, most schedulers use the backfilling mechanism. If the current highest-priority job cannot run because of insufficient processors, the scheduler regularly searches the waiting queue and allows short jobs to run ahead if they do not delay the higher-priority jobs (or at least the highest-priority jobs) [6, 7]. Obviously, the selection of backfilling job requires the knowledge of job runtime to fill the 'hole' [8]. Job runtime is an important foundation of the prioritizing policy and the backfilling mechanism.

With long-term operation of HPC platforms, more and more data are gathered in HPC systems by the hardware monitor (e.g., CPU usage, I/O traffic, and energy consumption), by the job management system, and by analytics at the application level (e.g., parameters, results, and temporary results) [9]. Maximizing the utilization of the accumulated data can effectively predict the runtime. However, previous studies and models have focused only on improving accuracy, resulting in higher computational complexity and time overhead, and difficulty in real-time scheduling system

^{*} This document is the results of the research project funded by the National Key Research and Development Program of China, grant number: 2016YFB0200902.

^{*}Corresponding author

 xjzhang@xjtu.edu.cn (X. Zhang)

ORCID(s):

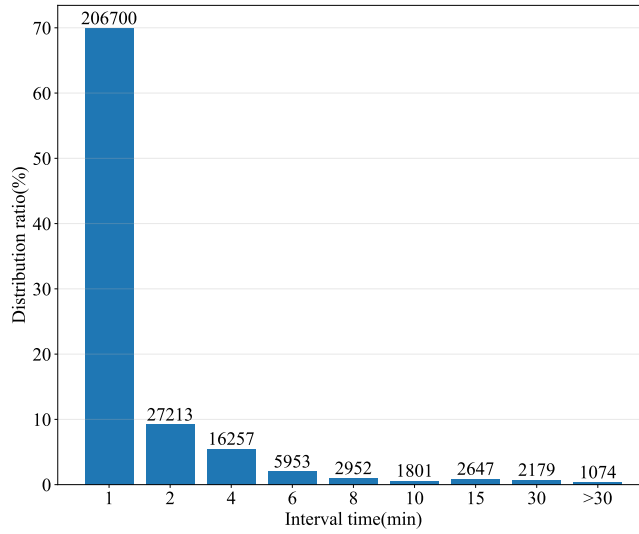


Figure 1: Time interval distribution of CEA-Curie-2011-2.1-cln workload.

deployment.

As shown in Figure 1, in the CEA-Curie-2011-2.1-cln workload [10], jobs with an arrival time interval that is less than 1 min account for 70% of the workload, up to 206,700 jobs. The update and reasoning time of the prediction model must be sufficiently short to meet the needs of real-time scheduling.

To address this problem, an efficient prediction model, referred to as online learning and k-nearest neighbors based (KNN-based) predictor with correction mechanism (OKCM), with low computational complexity and time overhead is proposed to predict job runtime for job scheduling. OKCM updates the model in real time through online learning with Huber loss function in low computational complexity and time overhead (less than 0.003s in our experiments). In addition, it is friendly to new users and users with a small amount of data through the KNN-based predictor, which is important for real systems. The correction mechanism is designed to ensure safe execution of high-priority and subsequent jobs.

To evaluate our model, we designed and implemented a trace-driven simulator, HPCsim, based on real job workloads. All simulated events and the related HPC environment including node resources, job scheduling, waiting queue, prioritizing policy, and backfilling mechanism are based on real workload traces and corresponding real HPC platform configurations. The results demonstrated that OKCM can achieve prediction accuracy of up to 82.3% with a low overhead. Experiments further demonstrate that this model can achieve significant scheduling performance improvements, can be used to enhance the main prioritizing and backfilling methods, and is not limited to a certain type of scheduling.

The remainder of this paper is organized as follows: Section 2 presents a review of the scheduling algorithm and closely related studies; Section 3 presents the proposed model to predict job runtime; Section 4 provides a demonstration of the accuracy of the proposed runtime prediction models and their scheduling performance; and Section 5 presents the conclusions and potential future research.

2. Related work

When a job arrives, the first step is to determine the priority of the job. The latest priority policy includes F1, F2, F3, F4 [3], FAT, WFP1, WFP3, and UNICEF [4]. Without exception, the runtime of the job is one of the essential input parameter.

To improve resource utilization, backfilling is currently the most common method in the scheduling system. Backfilling is a technique to improve resource utilization of HPC platforms by opportunistically running low-priority jobs when there are insufficient resources to run high-priority jobs [11]. In 1995, the seminal extensible Argonne schedul-

ing system (EASY) backfilling algorithm [12] was developed and is now the de facto standard scheduling algorithm in many mainstream resource managers, such as Slurm [13], a well-known workload manager. Since then, many backfilling variants have been proposed, such as slack-based backfilling supporting priorities and bounded wait times [14], or conservative backfilling, in which each waiting job is assigned a reservation [15]. To use backfilling, the scheduler must know the runtime of each job in advance for computing the reservation, to avoid delays in high-priority jobs.

In practice, most job management systems ask the user to estimate the runtime. However, runtimes supplied by users are acknowledged to be inaccurate and overestimated [11]. Users unfamiliar with the HPC platform and their own jobs cannot provide precise runtimes. They tend to significantly overestimate runtimes to avoid job termination for exceeding the allotted runtime. User-requested runtimes also embody an inherently harmful backfilling trend. Users tend to request round runtimes (e.g., 1 h), resulting in 90% of the jobs using the same 20% of runtime values [6], which limits the performance of backfilling in filling existing ‘holes’ because many jobs appear the same.

For some time, several researchers believed that inaccurate runtimes had a positive effect on backfilling as they provided more flexibility and led to better scheduling performance. It was even proposed that runtimes should be randomized [16]. In contrast, Tsafir et al. [6] demonstrated that precise knowledge of runtimes can have non-trivial effects on performance evaluation results.

Considerable research has been conducted to improve the accuracy of job runtime estimates. Tsafir et al. [6] found that using historical information from the same user had a better predictive effect than similar jobs. However, using a historical average cannot effectively capture dynamic runtime changes such as periodic characteristics. Gaussier et al. [5] used a polynomial model to improve scheduling performance through a weighted asymmetric loss function. This method focused on the final performance, and the prediction was biased. Yuping Fan et al. [2] reported that many current prediction methods improve prediction accuracy at the expense of increasing underprediction. They adopted the Tobit model’s censored regression capability to keep a low underestimation rate. Ju-Won Park et al. [9] adopted support vector regression (SVRegression) to estimate runtime. McKenna et al. [17] compared runtimes predicted by decision tree, random forest, and k-nearest neighbors, and indicated that the decision tree can achieve the best results. Thonglek et al. [18] improved resource utility using long short-term memory (LSTM). Goshgar Ismayilov et al. [19] used neural network based multi-objective evolutionary algorithm to improve scheduling. These studies and models focus only on improving accuracy, resulting in higher computational complexity and time overhead. They are difficult to deploy in a real-time scheduling system. Thus, an efficient prediction model that updates in real time and is friendly to new users is necessary.

3. OKCM prediction model for job scheduling

3.1. OKCM model

The problem researched is to execute a set of parallel jobs with rigid requested resources on large-scale HPC systems represented by a set of N_{max} homogeneous resources connected by interconnection topology. The jobs are submitted over time (online) to a centralized waiting queue. The job scheduler determines where and when to execute the jobs. Upon job completion, the system records the job along with a number of its attributes in the workload log. We define a *workload* with the following data:

- The identification (ID) of the job job_{id}
- The ID of the user $user_{id}$
- Number of compute nodes required by the User $node_{req}$
- Requested time t_{req} of the job by the user
- Wait time t_{wait} of the job (seconds)
- Submit time of the job t_{sub} (UTC timestamp, also known as release date and arrival time)
- Number of allocated nodes by the system $node_{sys}$
- Real runtime t_{real} of the job (seconds, only known after the job has been completed)
- Status s (1 = ok, 0 = fail, 5 = cancel)

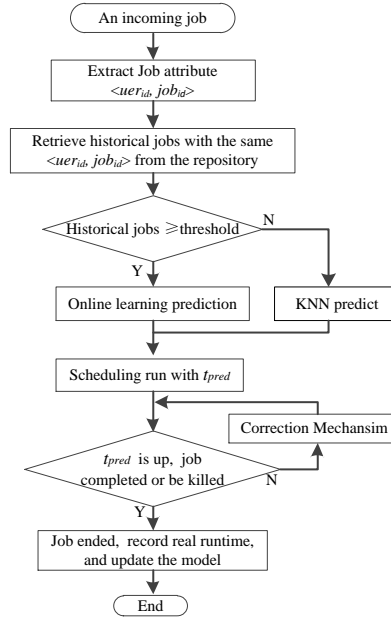


Figure 2: OKCM runtime prediction model.

Although some workload logs have additional and more detailed information, such as I/O, memory, and compilation information, our model requires only the most basic information, available in most workload traces, to establish a job repository. The workload logs are a subset of the standard workload format (SWF) [10]; thus, OKCM is applicable to a wide range of HPC platforms.

A high-level overview of OKCM is presented in Figure 2. Our time prediction model consists of two components, the prediction and correction mechanisms. Upon job arrival, OKCM retrieves job attributes from the repository. In the prediction mechanism, for users whose data accumulation is greater than the threshold (e.g., 25), online learning is used to predict the job runtime and update the model in real time. The KNN-based predictor is used for new users or users with less data than the threshold, which is important in a real system. During the job, effective runtime corrections are performed to guarantee safe execution of the job and ensure efficient scheduling of subsequent jobs.

3.2. Prediction mechanism

A *job* is represented by a vector $x \in \mathbb{R}^n$ where n is the number of features of the runtime prediction model, as shown in Table 1. These features are recorded in the job repository. All features describe the status of users and jobs from multiple dimensions and are contained in four categories:

- Characteristics of the current job to be predicted, extracted from submit command (e.g., $node_{req}, t_{req}$).
- Characteristics of user's job runtime, extracted from historical knowledge (e.g., $x_2, x_3, x_4, x_5, x_6, x_7, x_8$)
- Characteristics of nodes requirement of the user (e.g., x_9, x_{10}, x_{11})
- Characteristics of user behavior (e.g., x_{12}, x_{13}, x_{14})

For users whose data accumulation is greater than the threshold, *l2-regularized* online linear regression learning with the Huber loss function is proposed to provide predictions and perform updates in real time. The regression function is Equation 1:

$$f(w, x) = \sum_{i=0}^n w_i x_i \quad w_i \in \mathbb{R}^n \quad (1)$$

where w_i are the parameters to be learned, and x_i are the features of the *job*.

Table 1

Job features used for OKCM runtime prediction model.

Notation	Description	Meaning
x_0	number of compute nodes requested by the user $node_{req}$	characteristics of job to be predicted
x_1	requested time of the user t_{req}	
x_2	real runtime of the last job of the same user t_{real} , or t_{req} if it is the first job	characteristics of user's job runtime
x_3	real runtime of the second-to-last job of the same user t_{real2}	
x_4	average real runtime of the last two jobs of the same user AVG_{real2}	
x_5	real runtime of the third-to-last job of the same user t_{real3}	
x_6	average real runtime of the last three jobs of the same user AVG_{real3}	
x_7	sum of historical real runtimes for all jobs of the same user t_{all}	
x_8	average historical real runtime for all jobs of the same user AVG_{all}	characteristics of nodes requirement of the user
x_9	average historical requested nodes of the same user $node_{avgreq}$	
x_{10}	average historical real nodes of the same user $node_{avgreal}$	
x_{11}	$node_{req}$ normalized by average resource request	
x_{12}	sin value of the submit time of the week SIN_{week}	characteristics of user behavior
x_{13}	sin value of the submit time of the day SIN_{day}	
x_{14}	time interval since last job ended of the same user $t_{interval}$	

When training job runtime prediction model, the loss function is crucial. In workload files there are some abnormal jobs. Although their status is $ok(s = 1)$, they have particularly long or short runtimes. Therefore, the Huber loss function with $l2$ -regularized regression learning is suitable to reduce the effect of outliers on parameter estimation. In fact, the Huber loss function quadratically penalizes small errors to train efficiency, while the computation of absolute values of larger errors ensures regression robustness, as shown in Equation 2, where δ is a threshold.

$$L(f(w, x), t_{real}) = \begin{cases} \frac{1}{2} (f(w, x) - t_{real})^2 + \lambda \|w\|^2 & |f(w, x) - t_{real}| \leq \delta \\ \delta |f(w, x) - t_{real}| - \frac{1}{2} \delta^2 + \lambda \|w\|^2 & otherwise \end{cases} \quad (2)$$

In scheduling systems, whether the model satisfies real-time requirements is an important factor affecting the practicality of the model. In OKCM, the Huber loss and a normalized adaptive online gradient algorithm [20] are combined to train the model. It is proved that regret bounds are dependent on the ratio of scales existing in the data rather than the absolute scale. The data do not need to be pre-normalized; the infer-time and infer-complexity are reduced, and the algorithms are more robust. This is important for the scheduling system.

Compared to adaptive learning, when training, if the new features are larger than the previous ones, the weights are updated by the vector element s_i which stores the magnitude of feature i according to $s_i = \max |x_i|$. A gradient descent is performed according to Equation 3.

$$w_i = w_i - \eta \sqrt{\frac{t}{N}} \frac{1}{s_i \sqrt{G_i}} \frac{\partial L(f(w, x), t_{real})}{\partial w_i} \quad (3)$$

where,

$$N = N + \sum_i \frac{x_i^2}{s_i^2} \quad (4)$$

$$G_i = G_i + \left(\frac{\partial L(f(w, x), t_{real})}{\partial w_i} \right)^2 \quad (5)$$

Using N causes the learning rate to control the average change in prediction from an update. Using G completes the adaptive gradient descent.

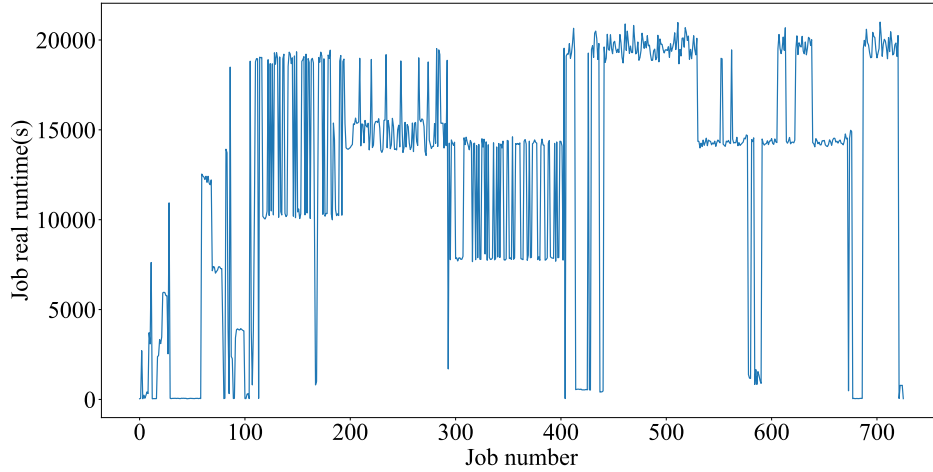


Figure 3: Real job runtime in ANL-Intrepid-2009-1 workload of user 176.

For new users or users with a small data accumulation, several machine learning algorithms, such as SVRegressor and light gradient boosting machine regressor (LGBMRegressor), which need huge amounts of data to learn, are unsuitable because of this inherent flaw in the algorithms. In our model, this problem is addressed using the KNN-based predictor. As shown in Figure 3, the overall runtime is bounded, and part of the time period is stepped. The running time of similar jobs in some time periods is similar. The lazy learning KNN is a simple but effective method to predict the runtime in job scheduling, shown as Algorithm 1. Only the runtime of the first job is set as t_{req} . The runtime of the second job is set as the first job runtime t_{first} . If the completed job number $2 \leq num \leq threshold$, the t_{pred} is learned by the KNN-based predictor.

Algorithm 1 KNN-based predictor

Input: Features $\langle x_0^j, x_1^j, x_{13}^j \rangle$ of all arrived job num extracted from job repository

Output: t_{pred} the predicted time of the job

- 1: **if** the new arriving job $job_{num} < threshold$ for the same user **then**
 - 2: Normalize all the features by its corresponding dimension $x_i = \frac{x_i - \bar{x}_i}{s_i}$
 - 3: $D^j \leftarrow$ normalized Euclidean distance $\sum_{i=0}^{num} (x_i - x_i^j)^2$
 - 4: Sort select the K neighbors by D^j and t_{sub}
 - 5: $t_{pred} \leftarrow$ the median runtime of the K neighbors
 - 6: **end if**
-

where, $\bar{x}_i = \frac{1}{n} \sum_{j=0}^{num} x_i^j$ and $s_i = \sqrt{\frac{1}{n-1} \sum_{j=0}^{num} (x_i - \bar{x}_i)^2}$ represent expected value and standard deviation, respectively. Features of the KNN-based predictor are a subset of the previous features, because in the same user space, the three features are able to characterize the similarity of jobs through normalized Euclidean distance. If the Euclidean distances are the same, the closest in time are k-nearest neighbors.

3.3. Correction mechanism

When predicting, the model may predict longer than the actual runtime (overprediction) or shorter than the actual runtime (underprediction). Overprediction is manageable, as it does not affect the running of the job. However, underprediction leads to unfinished jobs that must continue to run past the allotted runtime. For a deadline-constrained system, underprediction is technically unacceptable. Users do not tolerate their jobs being prematurely terminated simply because the model predicted a shorter runtime than the actual runtime.

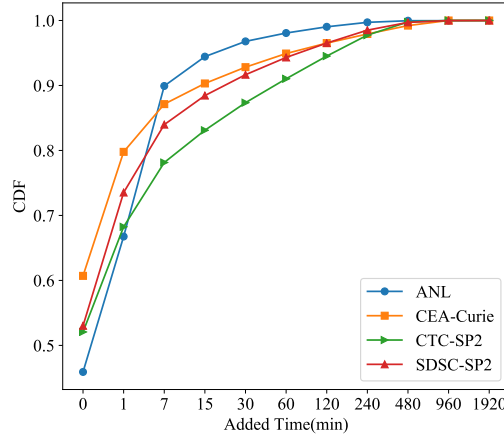


Figure 4: CFD result of incremental correction mechanism.

For underprediction, Tsafirir et al. [6] increased post-estimate predictions in a gradual manner. The first adjustment adds only one minute to the prediction. If this is not sufficient, the i^{th} prediction correction adds $15 \times 2^{i-2}$ min (15 min, 30 min, 1 h, 2 h, etc.), as shown in Figure 4. Such a method may allow more jobs to be backfilled, which may cause long wait times for high-priority jobs.

A checkpoint-based scheduling algorithm is used to solve such problem [21]. Jobs are checkpointed and terminated until the highest-priority job receives its requested resources. The checkpointed jobs are placed at the head of the queue to facilitate continued execution. However, this method requires additional system support.

Our model directly corrects the runtime to user-requested runtime. OKCM already has a high prediction accuracy, direct correction to the requested time can ensure the safe run of high-priority and subsequent jobs. The user-requested time serves as a kill-time. Correspondingly, the predicted time is only for the prioritizing policies and backfilling mechanism. Once the prediction is corrected, the model updates the remaining time and extra nodes to make new backfilling reservations.

3.4. HPCsim system

To evaluate the scheduling performance, HPCsim, a trace-driven simulator, was designed and implemented based on real system workload inputs.

All simulated events and the related HPC environment including node resources, job scheduling, job queue, prioritizing policy, and backfilling mechanism are based on real workload traces and corresponding real HPC platform configurations [10]. Prioritizing policies including F1, F2, F3, F4 [3], FAT, WFP1, WFP3, and UNICEF [4] are implemented in our scheduling system. The aggressive backfilling algorithm is optional with these priority policies. When using the backfilling option, jobs are sorted by priority, and the aggressive backfilling algorithm checks if low-priority jobs can be executed without delaying the highest priority jobs. All scheduling actions can be performed over the user-requested and predicted time. Upon a job termination, including a normal completion, the end of the predicted time, or a job being killed, the scheduler is notified to correct runtime or schedule other queued jobs on free nodes. SimGrid-3.25 [22] is used to realistically simulate the HPC environment and operation of HPC jobs with multiple nodes, computing speed, and network topology.

4. Experiment and analysis

4.1. Experiment scheme

Based on the simulator proposed in this study, a comprehensive verification scheme is designed, including prediction accuracy and scheduling performance. To ensure the validity of the accuracy test, walk-forward validation is used. When a job arrives, the current model is used to make a prediction. Once the actual runtime is known after the job is completed, the model is updated immediately for predicting the next job. This mimics a real-world job scheduling

scenario. To test the scheduling performance, each workload is sampled without replacement seven times, each time for 30 consecutive days. The median and mean values of average bounded slowdown and average waiting time are recorded and analyzed.

We analyzed four real workload traces collected from real HPC systems from the Parallel Workload Archive [10], including ANL-Intrepid-2009-1, CEA-Curie-2011-2.1-cln, CTC-SP2-1996-3.1-cln, and SDSC-SP2-1998-4.2-cln, as listed in Table 2. These traces are very representative, with configurations ranging from 128 to 163,840 nodes, jobs number from 73,496 to 321,826, and measurement dates from 1996 to 2011.

Table 2

Real workload traces used for evaluating runtime prediction and scheduling.

Name	Nodes	Job Total	Test Job range	Year	During
CEA-Curie-2011-2.1-cln	93,321	321,826	6,099-10,546	2011	20
ANL-Intrepid-2009-1	163,840	68,936	10,152-40,956	2009	8
CTC-SP2-1996-3.1-cln	338	77,222	5,985-8,391	1997	11
SDSC-SP2-1998-4.2-cln	128	73,496	2,189-3,523	1996	23

4.2. Prediction accuracy

The prediction accuracy is the ratio of t_{real} to t_{pred} , as shown in Equation 6.

$$accuracy = \begin{cases} 1 & t_{real} = t_{pred} \\ t_{pred}/t_{real} & t_{real} \geq t_{pred} \\ t_{real}/t_{pred} & t_{real} \leq t_{pred} \end{cases} \quad (6)$$

where t_{real} is the real runtime and t_{pred} is the predicted time by our model. An accuracy of 1.00 represents 100% accuracy. Figure 5 shows the prediction results of different algorithms. It is observed that the OKCM model achieved the best prediction accuracy on all workloads. The accuracy is 82.3% in ANL-Intrepid, 67.1% in CTC-SP2, 66.3% in SDSC-SP2, and 62.1% in CEA-Curie. OKCM prediction is 1.45-3.44 times more accurate than user-requested time. Compared with SVRegressor and LGBMRegressor algorithms, OKCM achieves a 3.3%-6.3% performance improvement.

4.3. Scheduling performance

Figure 6 shows the number of jobs scheduled for each workload. The performance is measured by the average bounded slowdown ($AVGblsd$) and average waiting time ($AVGwt$).

The $blsd$ slowdown of a job is the ratio of job response time to its actual runtime, which is defined as shown in Equation 7:

$$blsd^j = \max \left(\frac{t_{wait}^j + t_{real}^j}{\max(t_{real}^j, \tau)}, 1 \right) \quad (7)$$

where t_{wait}^j is the waiting time of job j and τ is a constant preventing overemphasis of the importance of extremely short jobs. In previous studies, τ has generally been set to 10 s. The $AVGblsd$ of M total jobs is defined as shown in Equation 8:

$$AVGblsd = \frac{1}{M} \sum_j \max \left(\frac{t_{wait}^j + t_{real}^j}{\max(t_{real}^j, \tau)}, 1 \right) \quad (8)$$

$AVGwt$ measures quality of service by the waiting time of M total jobs, defined as Equation 9.

$$AVGwt = \frac{1}{M} \sum_j t_{wait}^j \quad (9)$$

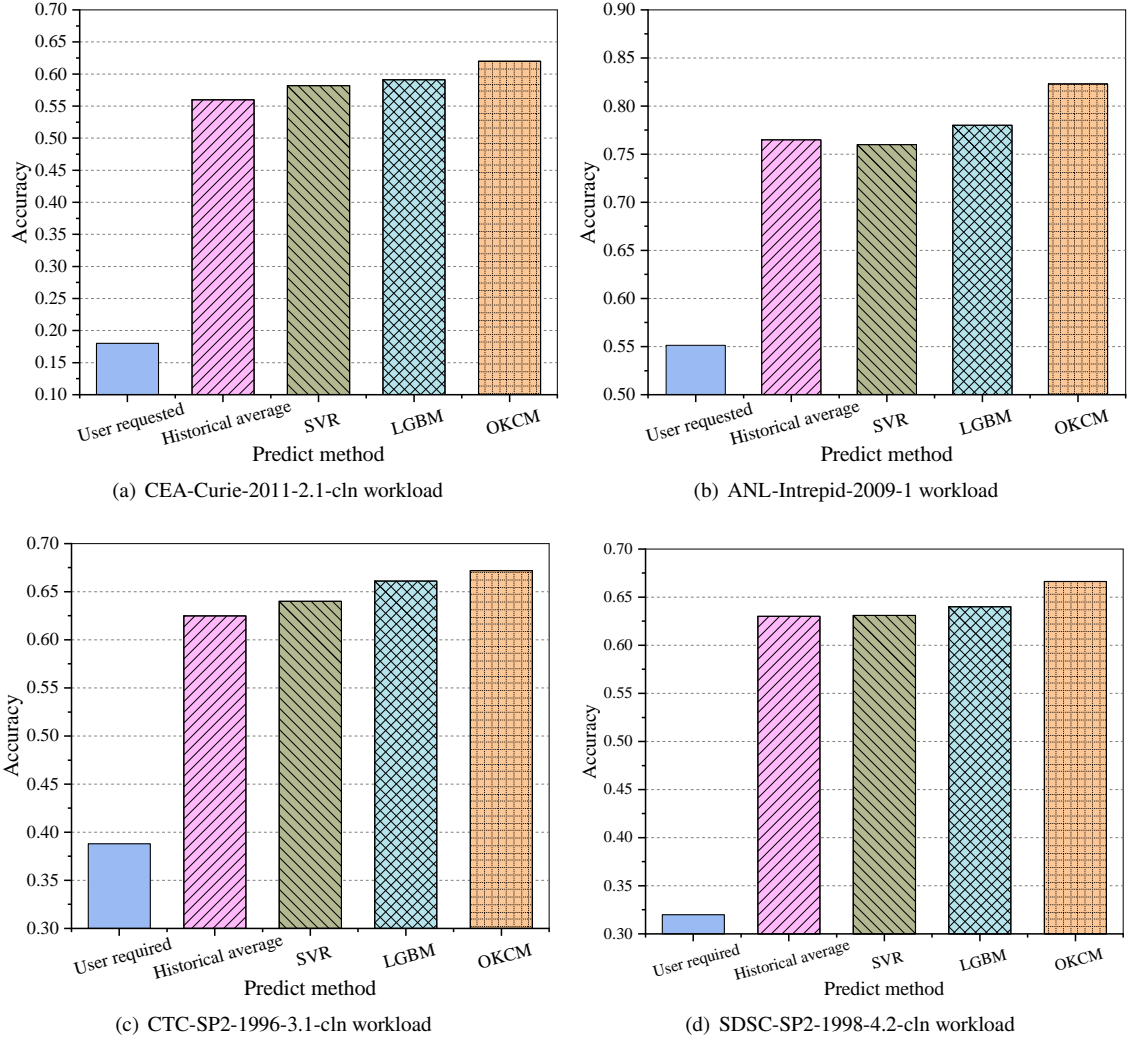


Figure 5: Comparison of runtime prediction accuracy of different algorithm.

The box-whisker plot is used to illustrate the scheduling performance in Figures 7-10. The orange line shows the median value, the box shows the 25th – 75th percentile values, and the red point is the mean of the data. In these figures, (a) shows the box plot of *AVGblsd* using user-requested time, (b) illustrates *AVGblsd* of OKCM predicted time, (c) and (d) show the result of *AVGwt* with user-requested time and OKCM predicted time, respectively.

4.3.1. CEA-Curie scheduling performance

Figure 7 shows the scheduling result of the CEA-Curie-2011-2.1-cln workload, and Tables 3-6 are the corresponding median and mean values of *AVGblsd* and *AVGwt* with user-requested time and predicted time, respectively.

Table 3

Median and mean *AVGblsd* scheduling performance with t_{req} of CEA-Curie workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	78.8	62.2	53.3	47.9	13.7	26.7	25.9	28.8
Mean	104.5	85.9	76.4	72.9	23.8	37.9	36.8	40.4

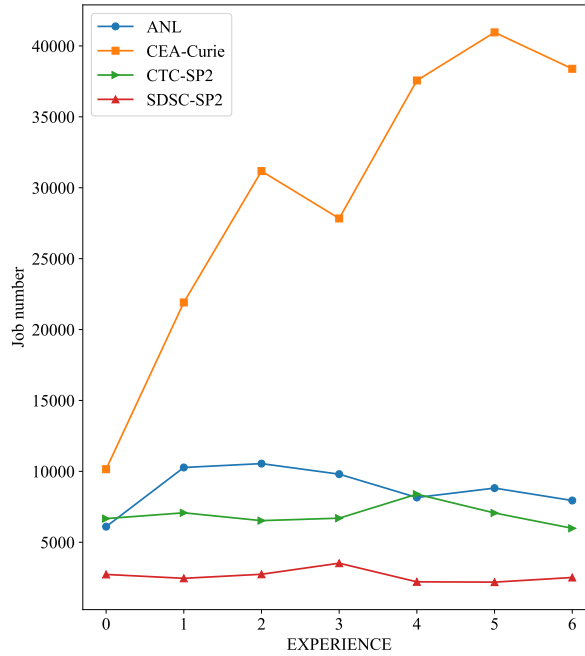


Figure 6: Job numbers for scheduling of each workload.

Table 4

Median and mean *AVGblsd* scheduling performance with t_{pred} of CEA-Curie workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	32.5	15.4	11.8	10.7	9.7	10.9	12.1	16.2
Mean	71.6	30.4	17.6	19.1	35.5	26.6	34.1	31.5

For the *AVGblsd* of CEA-Curie-2011-2.1-cln, shown as Table 3 and 4, except for the mean value of F4, all other scheduling performance is improved by OKCM according to Median and Mean indicators. For the median value, F4 scheduling obtains the optimal performance. In terms of mean value, UNI has the highest performance. UNI achieves the greatest improvement, 4.52 and 4.33 times better than user-requested time for the median and mean, respectively.

Table 5

Median and mean *AVGwt* scheduling performance with t_{req} of CEA-Curie workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	2935.3	2609.4	2067.2	1771.0	689.7	936.9	883.6	1070.2
Mean	3367.4	2740.2	2386.9	2296.9	1258.6	1329.5	1280.2	1397.2

For the *AVGwt* of CEA-Curie-2011-2.1-cln, shown as Table 5 and 6, all scheduling performance is improved by OKCM according to Median and Mean indicators. For the median value, F4 scheduling obtains the optimal performance, and UNI achieves up to 3.04 times improvement. In terms of the mean, SPT has the highest performance, and also achieves a maximum 2.72 times improvement. In addition, scheduling with our prediction runtime resulted in

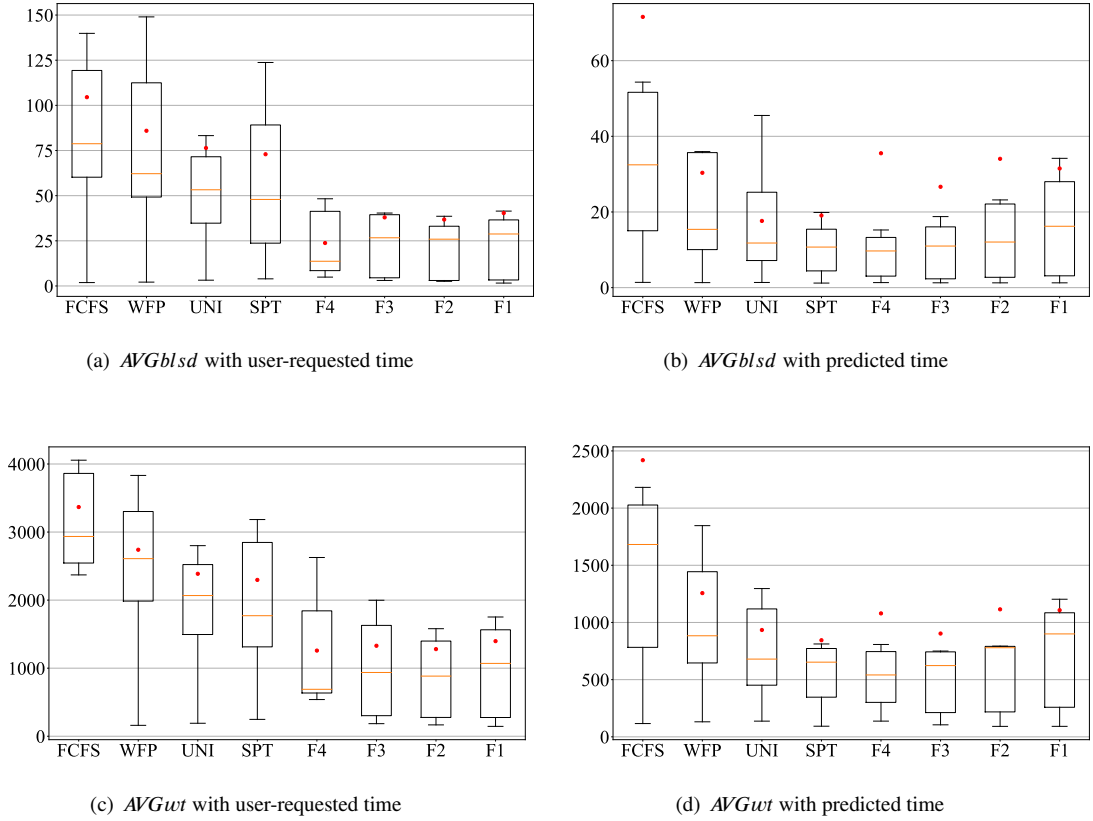


Figure 7: CEA-Curie workload scheduling performance.

Table 6

Median and mean *AVGwt* scheduling performance with t_{pred} of CEA-Curie workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	1681.9	883.7	679.7	652.9	541.1	622.9	779.2	899.9
Mean	2418.9	1256.6	934.3	844.7	1079.3	903.2	1114.8	1107.4

reduced differences between the 25% and 75% quartiles on both *AVGblsd* and *AVGwt* for the CEA-Curie-2011-2.1-cln workload.

4.3.2. ANL-Intrepid scheduling performance

Figure 8 shows the scheduling result of the ANL-Intrepid-2009-1 workload, and Tables 7-10 are the corresponding median and mean values of *AVGblsd* and *AVGwt* with user-requested time and predicted time, respectively.

Table 7

Median and mean *AVGblsd* scheduling performance with t_{req} of ANL-Intrepid workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	8.1	5.6	4.3	3.1	3.3	2.7	2.7	3.0
Mean	12.4	8.5	5.7	4.4	5.0	3.6	3.2	3.9

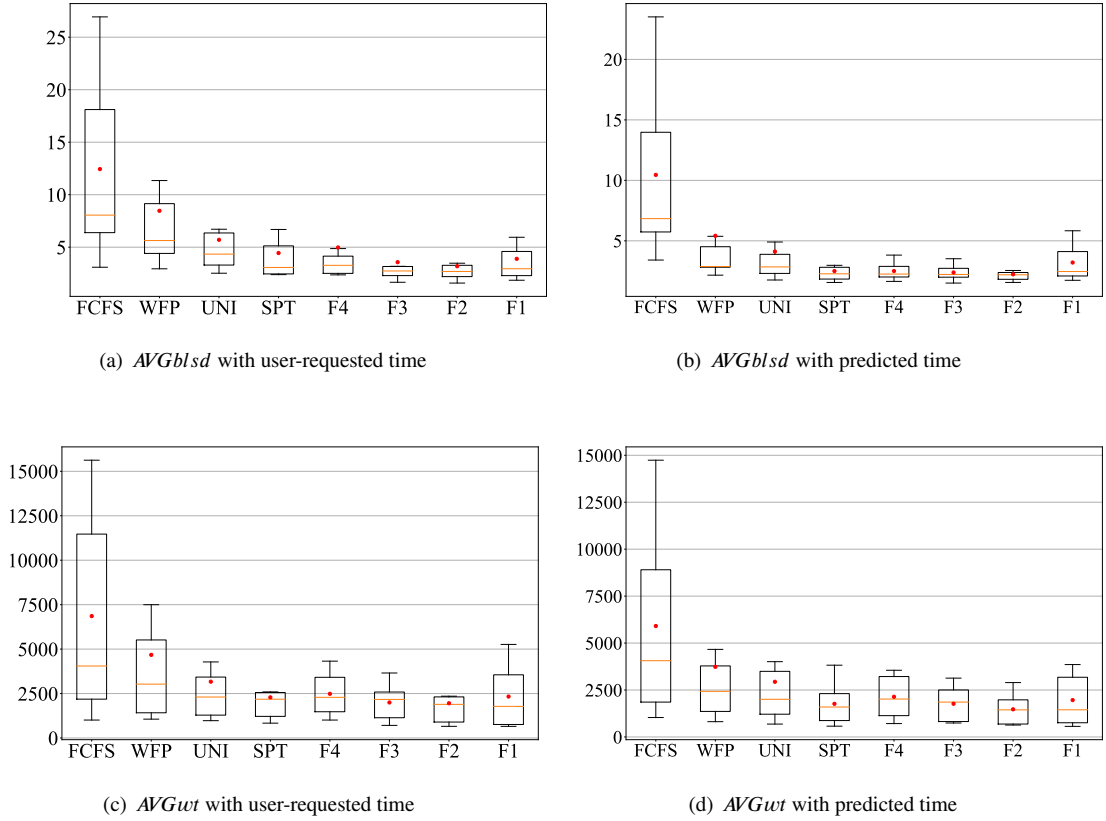


Figure 8: ANL-Intrepid workload scheduling performance.

Table 8

Median and mean *AVGblsd* scheduling performance with t_{pred} of ANL-Intrepid workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	6.8	2.9	2.8	2.3	2.3	2.2	2.1	2.5
Mean	10.5	5.4	4.1	2.5	2.5	2.4	2.2	3.2

For the *AVGblsd* of ANL-Intrepid-2009-1 workload, shown as Table 7 and 8, all scheduling performance is improved by OKCM for Median and Mean indicators. F2 obtains the optimal performance for both indicators. For the median value, WFP achieves up to 1.95 times improvement. In terms of mean, F4 achieves a maximum 1.99 times improvement.

For the *AVGwt* of ANL-Intrepid-2009-1, shown as Table 9 and 10, except for the median performance of FCFS (decreased by 0.3%), all other scheduling performance is improved by OKCM for both Median and Mean indicators. F2 obtains the optimal performance for both indicators. SPT achieved the greatest improvement, 1.37 and 1.30 times better than user-requested time for median and mean, respectively.

Scheduling with our prediction runtime resulted in reduced differences between the 25% and 75% quartiles on both *AVGblsd* and *AVGwt* for the ANL-Intrepid-2009-1 workload.

4.3.3. CTC-SP2 scheduling performance

Figure 9 shows the scheduling result of the CTC-SP2-1996-3.1-cln workload, and Tables 11-14 are the corresponding median and mean values of *AVGblsd* and *AVGwt* with user-requested time and predicted time, respectively.

Table 9

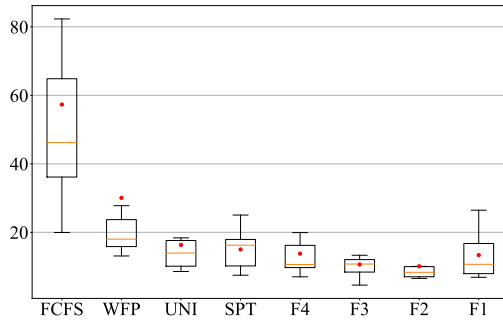
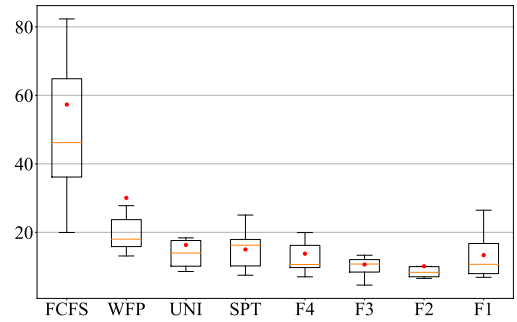
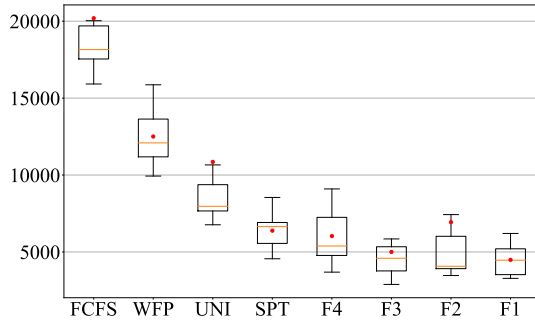
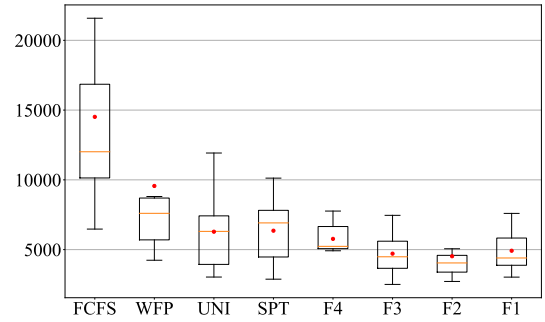
Median and mean *AVGwt* scheduling performance with t_{req} of ANL-Intrepid workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	4049.0	3029.7	2304.1	2181.2	2287.5	2169.4	1891.7	1778.1
Mean	6856.1	4676.9	3169.4	2285.8	2485.5	1997.7	1952.9	2331.9

Table 10

Median and mean *AVGwt* scheduling performance with t_{pred} of ANL-Intrepid workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	4061.9	2430.6	2006.7	1595.1	2018.8	1857.3	1443.3	1446.6
Mean	5908.3	3735.9	2939.2	1764.8	2141.2	1768.9	1471.9	1962.3

(a) *AVGblsd* with user-requested time(b) *AVGblsd* with predicted time(c) *AVGwt* with user-requested time(d) *AVGwt* with predicted time**Figure 9:** CTC-SP2 workload scheduling performance.

For the *AVGblsd* of CTC-SP2-1996-3.1-cln, shown as Table 11 and 12, except for the median performance of F3 (decreased by 15.1%), the median of F1 (decreased by 17.6), and the mean of F1 (decreased by 9.0), all other scheduling performance is improved by OKCM according to both indicators. F2 obtains the optimal performance for both indicators. For the median value, WFP achieves up to 2.43 times improvement. In terms of mean, UNI achieves a maximum 2.44 times improvement.

Table 11

Median and mean *AVGblsd* scheduling performance with t_{req} of CTC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	85.8	43.8	24.3	18.2	13.0	9.3	10.4	9.1
Mean	89.3	46.1	39.7	19.8	16.8	14.2	21.2	12.2

Table 12

Median and mean *AVGblsd* scheduling performance with t_{pred} of CTC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	46.2	18.0	13.9	16.2	10.6	10.7	8.3	10.7
Mean	57.3	30.1	16.3	15.0	13.8	10.6	10.1	13.3

Table 13

Median and mean *AVGwt* scheduling performance with t_{req} of CTC SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	18155.6	12089.5	7966.9	6646.6	5388.4	4589.8	4070.3	4465.1
Mean	20192.6	12504.5	10852.1	6385.1	6031.6	5002.1	6935.9	4487.9

Table 14

Median and mean *AVGwt* scheduling performance with t_{pred} of CTC SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	12012.9	7600.4	6306.8	6913.5	5233.2	4484.9	4043.9	4399.4
Mean	14515.1	9562.1	6282.8	6355.8	5768.8	4709.9	4523.9	4919.1

For the *AVGwt* of CTC-SP2-1996-3.1-cln, shown as Table 13 and 14, except for the median performance of SPT (decreased by 4.0%) and the mean of F1 (decreased by 9.6%), all other scheduling performance is improved by OKCM according to both indicators. F2 obtains the optimal performance for both indicators. For the median value, SPT achieves up to 1.59 times improvement. In terms of mean, UNI achieves a maximum 1.73 times improvement.

Scheduling with our prediction runtime resulted in reduced differences between the 25% and 75% quartiles in both *AVGblsd* and *AVGwt* for the CTC-SP2-1996-3.1-cln workload.

4.3.4. SDSC-SP2 scheduling performance

Figure 10 shows the scheduling result of the SDSC-SP2-1998-4.2-cln workload, and Tables 15-18 are the corresponding median and mean values of *AVGblsd* and *AVGwt* with user-requested time and predicted time, respectively.

Table 15

Median and mean *AVGblsd* scheduling performance with t_{req} of SDSC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	21.5	16.5	14.1	17.7	13.1	12.5	15.8	17.9
Mean	29.9	20.7	18.9	18.4	16.7	16.5	16.3	18.8

For the *AVGblsd* of SDSC-SP2-1998-4.2-cln, shown as Table 15 and 16, except for the mean performance of F4

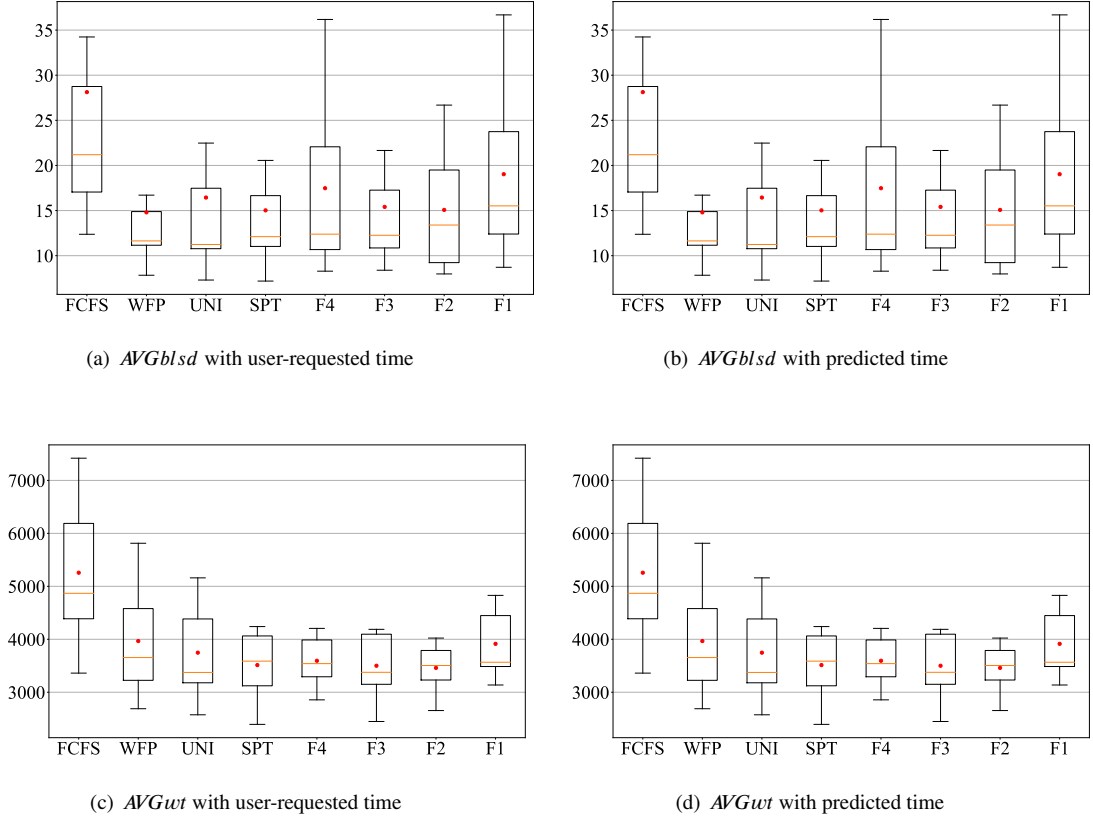


Figure 10: SDSC-SP2 workload scheduling performance.

Table 16

Median and mean $AVGblsd$ scheduling performance with t_{pred} of SDSC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	21.1	11.6	11.2	12.1	12.4	12.26	13.4	15.5
Mean	28.1	14.8	16.4	15.0	17.5	15.4	15.1	19.0

(decreased by 4.7%), all scheduling performance parameters are improved by OKCM according to both indicators. For the median value, UNI scheduling obtains the optimal performance, and SPT achieves up to 1.46 times improvement. In terms of mean, WFP has the highest performance, and also achieves a maximum 1.40 times improvement.

Table 17

Median and mean $AVGwt$ scheduling performance with t_{req} of SDSC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	5168.4	4135.5	3845.4	3406.9	3277.5	3391.3	3685.4	4415.9
Mean	6009.8	4853.5	4096.7	3622.7	3497.0	3472.7	3649.9	4246.3

For the $AVGwt$ of SDSC-SP2-1998-4.2-cln, shown as Table 17 and 18, except for the mean performance of SPT (decreased by 5.3%), the mean of F4 (decreased by 2.8%), and the mean of F3 (decreased by 0.8%), all other scheduling

Table 18

Median and mean *AVGwt* scheduling performance with t_{pred} of SDSC-SP2 workload.

Indicator	FCFS	WFP	UNI	SPT	F4	F3	F2	F1
Median	4868.3	3655.3	3373.9	3587.5	3540.8	3376.6	3504.9	3566.6
Mean	5257.4	3966.8	3747.7	3513.0	3594.7	3500.6	3460.3	3914.2

performance is improved by OKCM according to both indicators. For the median value, UNI scheduling obtains the optimal performance, and F1 achieves up to 1.24 times improvement. In terms of mean, F2 has the highest performance, and WFP achieves a maximum 1.22 times improvement.

Scheduling with our prediction runtime resulted in reduced differences between the 25% and 75% quartiles in both *AVGblsd* and *AVGwt* for the SDSC-SP2-1998-4.2-cln workload.

4.3.5. Overview of scheduling performance

Although the scheduling performance varied depending on the character of the jobs and the platform, the result was usually a significant improvement in performance.

Moreover, scheduling with required time produced either a large range of 25th – 75th percentile box, or outlier points with average slowdowns significantly higher than the median and mean, which damage the quality of service of the HPC system seriously. For all scheduling policy, scheduling with our prediction runtime resulted in reduced differences between the 25% and 75% quartiles in both *AVGblsd* and *AVGwt*. OKCM can be used to enhance the main prioritizing and backfilling methods and is not limited to a specific type of scheduling.

5. Conclusion

In this study, the OKCM model is proposed to improve the job runtime prediction for the HPC scheduling system. Previous studies and models focus only on improving accuracy, resulting in higher computational complexity, time overhead, and difficult deployment in real-time scheduling systems. Our design updates the model in real time through online learning and is friendly to new users or users with a small amount of data through the KNN-based predictor, which is an important feature for a real system. The correction mechanism is designed to guarantee the safe running of high-priority and subsequent jobs. The trace-driven simulator HPCsim was designed implemented based on real job workloads to evaluate the model. The results demonstrated that OKCM led to important scheduling performance improvements and enhanced primary prioritizing and backfilling methods without scheduling limitations.

In future research, we would improve in two directions. First, we would deploy the model using real HPC platforms and assess performance with real jobs and users. We used a realistic simulation to evaluate the prediction model and its scheduling performance using real traces. However, the traces ignored some important factors, such as memory bottlenecks that could affect job execution. Nevertheless, the simulations are a good approximation of these HPC platforms.

The second direction would be research in a wider range of HPC platforms, containing computing units with distinct architectures such as GPUs [23] and the Sunway TaihuLight-like supercomputer [24]. We plan to continue optimizing the OKCM model to improve scheduling performance of heterogeneous HPC platforms.

We recommend that when a job is completed, the system pushes the real runtime of the job to the user, so the user can compare the time and provide a more accurate requested time when submitting the next job. This feature has been implemented in our simulator and is found to yield a significantly improved accuracy for user-supplied time.

6. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

7. Acknowledgments

This research was funded by the National Key Research and Development Program of China, grant number: 2016YFB0200902.

References

- [1] Éric Gaussier, Jérôme Lelong, Valentin Reis, and Denis Trystram. Online tuning of easy-backfilling using queue reordering policies. *IEEE Trans. Parallel Distrib. Syst.*, 29(10):2304–2316, 2018.
- [2] Yuping Fan, Paul Rich, William E. Allcock, Michael E. Papka, and Zhiling Lan. Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*, pages 530–540. IEEE Computer Society, 2017.
- [3] Danilo Carastan-Santos and Raphael Y. de Camargo. Obtaining dynamic scheduling policies with simulation and machine learning. In Bernd Mohr and Padma Raghavan, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017*, pages 32:1–32:13. ACM, 2017.
- [4] Wei Tang, Zhiling Lan, Narayan Desai, and Daniel Buettner. Fault-aware, utility-based job scheduling on blue, gene/p systems. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA*, pages 1–10. IEEE Computer Society, 2009.
- [5] Éric Gaussier, David Glessner, Valentin Reis, and Denis Trystram. Improving backfilling by using machine learning to predict running times. In Jackie Kern and Jeffrey S. Vetter, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 64:1–64:10. ACM, 2015.
- [6] Dan Tsafir, Yoav Etsion, and Dror G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, 2007.
- [7] Mohammad Abu Obaida and Jason Liu. Simulation of HPC job scheduling and large-scale parallel workloads. In *2017 Winter Simulation Conference, WSC 2017, Las Vegas, NV, USA, December 3-6, 2017*, pages 920–931. IEEE, 2017.
- [8] Dan Tsafir, Yoav Etsion, and Dror G. Feitelson. Modeling user runtime estimates. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing, 11th International Workshop, JSSPP 2005, Cambridge, MA, USA, June 19, 2005, Revised Selected Papers*, volume 3834 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2005.
- [9] Ju-Won Park and Eunhye Kim. Runtime prediction of parallel applications with workload-aware clustering. *J. Supercomput.*, 73(11):4635–4651, 2017.
- [10] Dror G. Feitelson, Dan Tsafir, and David Krakov. Experience with using the parallel workloads archive. *J. Parallel Distributed Comput.*, 74(10):2967–2982, 2014.
- [11] Dalibor Klusáček and Václav Chlumský. Evaluating the impact of soft walltimes on job scheduling performance. In Dalibor Klusáček, Walfredo Cirne, and Narayan Desai, editors, *Job Scheduling Strategies for Parallel Processing - 22nd International Workshop, JSSPP 2018, Vancouver, BC, Canada, May 25, 2018, Revised Selected Papers*, volume 11332 of *Lecture Notes in Computer Science*, pages 15–38. Springer, 2018.
- [12] Joseph Skovira, Waiman Chan, Honbo Zhou, and David A. Lifka. The EASY - loadleveler API project. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, IPPS'96 Workshop, Honolulu, Hawaii, USA, April 16, 1996, Proceedings*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer, 1996.
- [13] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: simple linux utility for resource management. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003.
- [14] David Talby and Dror G. Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *13th International Parallel Processing Symposium / 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP '99), 12-16 April 1999, San Juan, Puerto Rico, Proceedings*, pages 513–517. IEEE Computer Society, 1999.
- [15] Ahuva Mu'alem Weil and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.
- [16] Dejan Perkovic and Peter J. Keleher. Randomization, speculation, and adaptation in batch schedulers. In Jed Donnelley, editor, *Proceedings Supercomputing 2000, November 4-10, 2000, Dallas, Texas, USA. IEEE Computer Society, CD-ROM*, page 7. IEEE Computer Society, 2000.
- [17] Ryan McKenna, Stephen Herbein, Adam Moody, Todd Gamblin, and Michela Taufer. Machine learning predictions of runtime and IO traffic on high-end clusters. In *2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016*, pages 255–258. IEEE Computer Society, 2016.
- [18] Kundjanasith Thonglek, Kohei Ichikawa, Keichi Takahashi, Hajimu Iida, and Chawanat Nakasan. Improving resource utilization in data centers using an lstm-based prediction model. In *2019 IEEE International Conference on Cluster Computing, CLUSTER 2019, Albuquerque, NM, USA, September 23-26, 2019*, pages 1–8. IEEE, 2019.
- [19] Goshgar Ismayilov and Haluk Rahmi Topcuoglu. Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. *Future Gener. Comput. Syst.*, 102:307–322, 2020.
- [20] Stéphane Ross, Paul Mineiro, and John Langford. Normalized online learning. In Ann Nicholson and Padhraic Smyth, editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press, 2013.
- [21] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Mingliang Liu, Yan Zhai, Wenguang Chen, and Weimin Zheng. Employing checkpoint to improve job scheduling in large-scale systems. In Walfredo Cirne, Narayan Desai, Eitan Frachtenberg, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing, 16th International Workshop, JSSPP 2012, Shanghai, China, May 25, 2012. Revised Selected Papers*, volume 7698 of *Lecture Notes in Computer Science*, pages 36–55. Springer, 2012.

- [22] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel Distributed Comput.*, 74(10):2899–2917, 2014.
- [23] Cong Thuan Do, Hong Jun Choi, Sung Woo Chung, and Cheol Hong Kim. A novel warp scheduling scheme considering long-latency operations for high-performance gpus. *J. Supercomput.*, 76(4):3043–3062, 2020.
- [24] Jingbo Li, Xingjun Zhang, Jianfeng Zhou, Xiaoshe Dong, and Chuhua Zhang. swHPFM: Refactoring and optimizing the structured grid fluid mechanical algorithm on the sunway taihulight supercomputer. *Appl. Sci.*, 10(1):72, 2020.

CRediT authorship contribution statement

Jingbo Li: Conceptualization of this study, Methodology, Software, Investigation, Data curation, Writing-Original draft preparation, Writingreview and editing, Validation. **Xingjun Zhang:** Formal analysis, Writing-review and editing, Project administration, Validation, Resources, Supervision, Funding acquisition. **SomeOne1:** Validation, Supervision. **SomeOne2:** Validation, Supervision.



Jingbo Li is currently a Ph.D. candidate in School of Computer Science and Technology at Xi'an Jiaotong University. He is a student member of CCF. His main research interests include computer architecture, high performance computing, parallel scheduling and machine learning.



Xingjun zhang is currently a Professor in School of Computer Science and Technology at Xi'an Jiaotong University. He is a member of CCF. His research interests mainly include computer architecture, high performance computing, big data storage system and computer networks.