Complex & Intelligent Systems (2023) 9:5731–5744
https://doi.org/10.1007/s40747-023-01045-z

ORIGINAL ARTICLE

A time-sensitive learning-to-rank approach for cloud simulation resource prediction

Yuhao Xiao[1] · Yiping Yao[1] · Kai Chen[1] · Wenjie Tang[1] · Feng Zhu[1]

Received: 7 October 2022 / Accepted: 9 March 2023 / Published online: 3 April 2023

Abstract
Predicting the computing resources required by simulation applications can provide a more reasonable resource-allocation scheme for efficient execution. Existing prediction methods based on machine learning, such as classification/regression, typically must accurately predict the runtime of simulation applications and select the optimal computing resource allocation scheme by sorting the length of the simulation runtime. However, the ranking results are easily affected by the simulation runtime prediction accuracy. This study proposes a time-sensitive learning-to-rank (LTR) approach for cloud simulations resource prediction. First, we use the Shapley additive explanation (SHAP) value from the field of explainable artificial intelligence (XAI) to analyze the impact of relevant factors on the simulation runtime and to extract the feature dimensions that significantly affect the simulation runtime. Second, by modifying the target loss function of the rankboost algorithm and training a time-sensitive LTR model based on simulation features, we can accurately predict the computing resource allocation scheme that maximizes the execution efficiency of simulation applications. Compared with the traditional machine learning prediction algorithm, the proposed method can improve the average sorting performance by 3%–48% and can accurately predict the computing resources required for the simulation applications to execute in the shortest amount of time.

Keywords Cloud computing · Complex system simulation · Computing resource prediction · Learning to rank

Introduction

Simulations of complex systems are widely used in the defense, energy, transportation, aerospace, and other fields. Modeling and simulation technology is one of the most important methods for studying complex systems. Higher requirements have been proposed for the computing and communication capabilities of simulation systems because of the growing computational load and the runtime of complex system simulation applications [1]. Parallel discrete event simulations (PDES) are often used to improve the efficiency of complex system simulation applications. It is typically necessary to specify the amount of computing resources (referred to in this study as the number of CPU cores) required for the complex system simulation applications based on PDES before they are executed and divide the simulation task into different CPU cores for parallel computation [2]. Cloud computing technology can realize collaborative management and on-demand distribution of computing/storage and other resources to provide efficient resource allocation tools for complex system simulation applications based on parallel and discrete simulation (PADS) [3]. However, a high communication delay occurs in the cloud environment because computing and storage resources are often distributed among different computing nodes [4]. During the execution of the simulation task, communication is frequently synchronized, and numerous messages are sent and received. The execution efficiency of the simulation application in the cloud environment will be affected if there is an excessive allocation of computing resources, which will increase communication overhead between the simulation entities [5]. Therefore,

✉ Feng Zhu
zhufeng@nudt.edu.cn

Yuhao Xiao
xiaoyuhao19@nudt.edu.cn

Yiping Yao
ypyao@nudt.edu.cn

Kai Chen
chenkai@nudt.edu.cn

Wenjie Tang
tangwenjie@nudt.edu.cn

1 College of Systems Engineering, National University of Defense Technology, Changsha 410073, China

a method is required to predict the priority order of the simulation runtime corresponding to the number of different simulation computing resources before the simulation application is executed and to reasonably allocate computing resources for the simulation application according to the priority order.

Several researchers have recently investigated classification/regression machine learning-based simulation resource prediction methods in cloud environments [6,7]. These methods collect the characteristic data of simulation applications and use classification/regression algorithms to predict the runtime of simulation applications using various computing resources. Finally, these methods rank simulation computing resources based on the prediction results (simulation runtime) and expect to obtain those that will enable the simulation applications to execute in the shortest time. However, these methods typically base their prediction models on high-precision prediction indicators, such as root mean square error (RMSE) [8] or mean absolute percentage error (MAPE) [9]. The optimization objective is to minimize the difference between the predicted and actual runtimes, which makes the simulation's ordination susceptible to prediction accuracy. Compared with the prediction method that directly optimizes the overall ordination accuracy of computing resources, a method based on the prediction before ordination requires a higher time and space computational complexity and makes it difficult to achieve higher sorting accuracy. The learning-to-rank (LTR) method uses the relative ranking between pairwise samples in the dataset as feedback information and reduces the error ranking probability between sample pairs through iterative learning to produce the best ranking possible [10]. Therefore, during the task of simulation computing resource prediction, the method that directly learns the ranking of resource usage can explain the relationship between different resource usage and simulation application runtime at a deeper level than the method that predicts the simulation runtime under different computing resources and re-ranking.

To address the above problems, this study proposes a time-sensitive LTR approach for resource prediction in cloud simulation. The method collects static feature and dynamic monitoring data from simulation applications, and analyzes the impact of various factors on the execution efficiency of the simulation based on the feature extraction method of SHapley additive exPlanation (SHAP) interpretation framework. Second, based on the analysis of the factors that significantly affect the runtime of simulation applications, the objective loss function of the proposed method was designed, and the RankBoost model was used to construct the simulation runtime-sensitive predictor. The method uses the shortest simulation runtime as the training goal and adds a time penalty factor to the loss function of the LTR model to ensure that more penalty is added when the best resources are not correctly ranked to accurately predict the computing resources with the shortest simulation runtime.

The main contributions of this study are as follows:

1. This study used the feature extraction method of the SHAP interpretability framework to quantitatively evaluate the importance of factors that influence the runtime of a complex system simulation. First, the set of factors that affect the efficiency of the simulation was analyzed, and the static characteristics before the execution of the simulation application and the dynamic characteristics during the dynamic execution of the simulation application were extracted as the feature data of the training samples. Second, the significant contribution of each feature was calculated based on the SHAP value, and the features that significantly affect the simulation application's runtime are extracted as input to improve the performance of the predictor.
2. This study proposes a time-sensitive learning-to-rank approach for resource prediction in cloud simulation. The method designs the target loss function of the RankBoost algorithm and trains a time-sensitive predictor based on the simulation features to accurately predict the computing resource allocation scheme that maximizes the execution efficiency of simulation applications. Compared with other eight machine learning methods, the experimental results show that our method can achieve the shortest simulation runtime and improve the accuracy of the optimal computing resource ranking.

Related work

Several studies have been conducted on the acceleration technology of complex system simulation applications combined with the advantages of on-demand sharing and elastic scaling in cloud computing environments. However, although cloud computing-based simulation technology has several advantages, the random resource allocation process in the cloud environment may also reduce simulation performance [11]. To address this problem, regression/classification prediction models based on machine learning, such as linear regression [12,13], nearest neighbor [14], regression tree [15], support vector machine [16], Bayesian [17], neural networks [18,19], and ensemble learning models [20,21], are the widely accepted solutions to this problem. These methods attempt to accurately predict the dynamic properties (such as computing resource requirements, runtime, or load fluctuations) of simulation applications using deep learning in many historical feature datasets.

Bin [22] modeled computational resource planning in cloud environments as a classification problem using a weighted support vector machine (SVM) model to predict future workload changes, thus minimizing the cost of cloud resource allocation and virtual machine provisioning. To overcome the limitations of reactive cloud autoscaling, Kee proposed an online framework for cloud computing resource prediction using multiple predictors to create an ensemble model for accurate prediction of actual workloads [6]. To improve the responsiveness of cloud management systems and improve the quality of cloud services, Nawrocki [23] combined anomaly detection and machine learning methods to allocate cloud resources to reduce cost and improve the quality of service (QoS). To cope with unexpected events in the cloud environment, Shen [24] proposed an efficient method for predictive resource allocation in the cloud. The approach uses machine learning models to predict contingencies and normal events and execute different resource allocation schemes. To improve the allocation efficiency of virtual machines in heterogeneous cloud environments, Mahfoudh [25] used a diffusion convolutional recurrent neural network model to predict the fluctuation of computational load in the next time period in real time and validated the model on a real-world dataset of CPU usage traces from PlanetLab. For time-series data in cloud computing environments, Jing [26] integrated bidirectional and lattice-based long- and short-term memory network models to achieve high-quality prediction of workload and resource usage in cloud environments.

The above machine learning methods typically aim to build a prediction model while improving the absolute accuracy (that is, reducing the RMSE, MAE, and other indicators). To obtain an effective resource ranking, these methods must predict the simulation runtime under various resource conditions and then rank the various resources according to the prediction results. However, compared with learning-to-rank method that directly optimize the overall ranking, classification/regression models that indirectly rank based on absolute accuracy tend to obtain poor ranking results [27].

Compared with classification/regression-based prediction methods, research on ranking computing resources for simulation applications using the LTR method is still very limited. Currently, most mainstream LTR algorithms adopt the pairwise approach, which considers a sample pair as a single training sample and focuses on the relative order of two samples rather than the specific value of a single sample. Some common methods include RankSVM [28], RankBoost [29], RankNet [?], and others [30,31]. Nguyen [32] empirically demonstrated using five datasets that the ranking accuracies of two LTR methods (RankSVM and RankBoost) were 4%–21% higher than that of the liner regression model (LR). Yang [33] proposed an LTR method to develop a software defect prediction model by directly optimizing the ranking performance. The method's accuracy in ranking the metrics was demonstrated. Jacky [27] analyzed the cost and data imbalance problems, and proposed a defect prediction method based ranking SVM model to improve the accuracy of software defect prediction. Burges et al. [34] proposed an ensemble prediction algorithm that combines the LambdaMART and LambdaRank algorithms and demonstrated through experimental datasets that the algorithm produced better prediction results.

The LTR algorithm of AdaBoost selects a feature variable each time to learn the weak sorters, which are then combined to form a strong sorter. The RankBoost algorithm inherits the advantages of the AdaBoost algorithm and has a fast convergence speed and strong generalization ability [35]. Therefore, this study combines the advantages of the RankBoost algorithm and proposes a time-sensitive LTR approach for resource prediction in cloud simulation. The approach selects the number of features that significantly affect the simulation efficiency based on the SHAP interpretability framework and then designs the target loss function of the RankBoost algorithm to train a time-sensitive LTR model. Compared with other LTR models, the proposed model adds more penalties when the ranking of the best resources is incorrect.

The time-sensitive learning-to-rank approach for resource prediction

Cloud computing technology could greatly improve the efficiency of resource allocation for complex system simulation applications [36]. Compared with the traditional method of using classification/regression technology to indirectly predict the use of computing resources for simulation applications, the LTR method can better explain the relationship between different resource usages and the execution efficiency. Therefore, this study proposes a time-sensitive learning-to-rank approach (TSLTR) to support optimal resource allocation for simulation applications. The fundamental concept is to use an LTR algorithm to predict the priority of the computing resources used before a simulation application is executed.

This method aims to reduce the number of resource pairs with incorrect rankings, obtain the highest ranking of resource usage, and improve the execution efficiency of simulation applications. In practice, the PADS application focuses more on resource usage, which speeds up the simulation application's execution. Therefore, this study adds a penalty factor to the loss function of the model to improve the ranking accuracy of top-ranked resources. The architecture of the proposed approach is shown in Fig. 1

During the initial phase of the approach, we deployed resource monitors on cloud computing nodes to collect real-time operational information of the emulated applications. In this study, we considered two sets of simulation parameters:

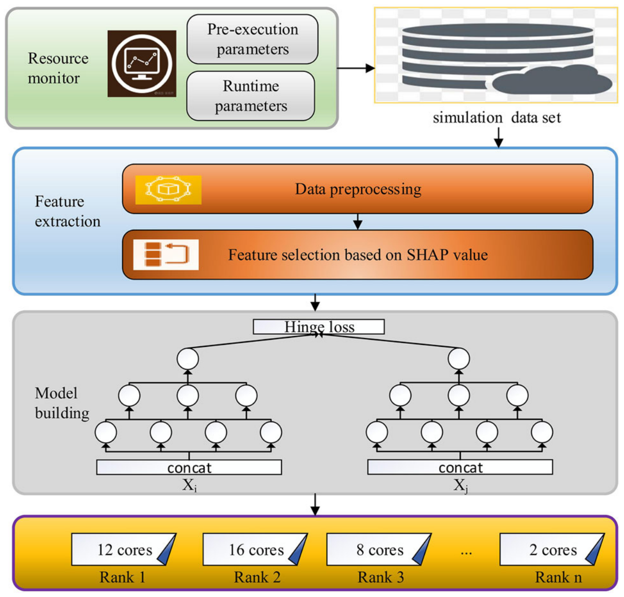


Fig. 1 Architecture of the proposed approach

pre-execution and runtime parameters. The pre-execution parameters, which reflect the static characteristics of the simulation application, are set before the application is executed. The runtime parameters are collected while the simulation application is being executed and reflect the performance differences under different cloud computing resources. The resource monitor collects information every 3 s and deposits it into the cloud application database with the specific parameters shown in Table 1. During the second stage, the collected dataset was processed using feature extraction methods to obtain relevant features. First, data preprocessing and data cleaning methods were performed to remove the null values and negative values. Then, standardization was used to normalize all data to a distribution with a mean of 0 and a variance of 1 so that eliminating the negative effects of abnormal samples. Second, based on the SHAP explainable framework, we analyzed the impact of each feature dimension on the operational efficiency of the simulation application and selected strongly relevant features to reduce the model error and achieve a more accurate prediction result. Finally, the extracted relevant features were used as inputs to train the LTR model to predict the priority order of various cloud resource usage. The highest ranked resource was used as the final prediction result.

The LTR model

The LTR algorithm aims to use the data from the previous execution to predict the priority of the computing resource usage for the next execution. This method uses the relative ranking between sample pairs in the training sample set as feedback information and reduces the probability of incorrect ranking between sample pairs by iteratively learning to produce the best ranking for all samples. Therefore, this study uses the LTR algorithm to predict the overall ranking of computing resources as well as to provide an optimal resource allocation scheme for achieving the shortest simulation runtime.

(1) Problem description

Consider the ranking objective named $SetX$, which ranks a set of applications S_i with the same simulation static parameters (except for the CPU cores), where X represents the CPU cores assigned to the simulation execution. The optimal resource requirements for cloud-based simulation applications can be expressed as $P_i = f\,(S_i(x_i, y_i))$, where $x_i = (x_1, x_2, \ldots x_d)$ represents the d-dimensional feature vector of the simulation application, y_i represents the simulation runtime, and f denotes the ranking function. The LTR model aims to learn from the dataset to obtain a sorting function that correctly sorts the number of resources to be used. Therefore, the feedback function is defined as follows:

$$SetX = \varphi\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) = \begin{cases} +1 & S_j\left(x_j, y_j\right) > S_k\left(x_k, y_k\right) \\ 0 & S_j\left(x_j, y_j\right) = S_k\left(x_k, y_k\right) \\ -1 & S_j\left(x_j, y_j\right) < S_k\left(x_k, y_k\right) \end{cases} \tag{1}$$

For any pair of samples S_j and S_k in any set of applications, $S_j\left(x_j, y_j\right) > S_k\left(x_k, y_k\right)$ indicates that the simulation runtime when using P_j resources is higher than the simulation runtime when using P_k resources. $S_j\left(x_j, y_j\right) < S_k\left(x_k, y_k\right)$ implies that the simulation runtime when using P_j resources is lower than that when using P_k resources. $S_j\left(x_j, y_j\right) = S_k\left(x_k, y_k\right)$ indicates that the simulation runtime when using P_j resources is equal to that when using P_k resources. The above LTR task was converted to a binary classification task that involves learning the relative order between sample pairs. Based on this, a weight D is assigned to each sample pair, indicating the importance of accurately judging the sample pairs. The distribution of D is defined as follows:

$$D\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) = c \cdot \max\left\{0, \varphi\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right)\right\}, \tag{2}$$

where c represents a positive constant and satisfies the following:

$$\sum_{S_j, S_k} D\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) = 1. \tag{3}$$

Therefore, the LTR algorithm aims to use positive feedback sample pairs with weights D to learn a final ranking function H. The probability of a ranking error between the

Table 1 Description of the monitoring parameters

Parameters	Indicators	Description
Pre-execution parameters	CPU core	Number of CPU cores used
	Entity	Number of simulation entities
	Events	Number of simulation events executed
	Lookahead	Restricts the simulation events that can be executed between the current timestamp and the sum of timestamp and lookahead values
	Simulation time	The virtual time in simulation applications
Runtime parameters	CPU usage	Average CPU usage
	CPU usage_max	CPU usage maximum
	CPU usage_min	CPU usage minimum
	Memory	Average memory consumption
	Memory_max	Memory consumption maximum
	Memory_min	Memory consumption maximum
	File	Hard drive read/write capacity
	Network rec	Network upload rate
	Network rec_max	Maximum network upload rate
	Network rec_min	Minimum network upload rate
	Network sent	Network download rate
	Network sent_max	Maximum network download rate
	Network sent_min	Minimum network download rate
	Network delay	Network latency

best ranking and the predicted ranking in the learning process is called the ranking loss (Rloss), and its utility function $Func$ and loss function $Rloss$ are defined as follows:

$$Func = \sum_{S_j, S_k} D_r\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) \times \left(h_r\left(S_j\left(x_j, y_j\right)\right) - h_r\left(S_k\left(x_k, y_k\right)\right)\right) \tag{4}$$

$$\text{Rloss} = \sum_{s_j, S_k} D_r\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) \times \exp\left(\alpha_r\left(h_r\left(S_j\left(x_j, y_j\right) - h_r S_k\left(x_k, y_k\right)\right)\right)\right) \tag{5}$$

where r denotes the number of iterations, α_r denotes the weight of the weak classifier obtained in the current iteration, and h_r denotes a function with 0–1 values. $\alpha_r > 0$ indicates that the ranking effect of h_r is positive, that is, the correct rate is more than half. Therefore, by selecting the appropriate α_r and h_r in each weak learning process, we can reduce the Rloss of the final ranking result. In this study, we set the α_r weight value by referring to [24,30], defined as follows:

$$\alpha_r = 0.5 \ln\left(1 + r(h)_{\max}\right) / \left(1 - r(h)_{\max}\right). \tag{6}$$

(2) Algorithm description

The specific execution steps are shown in Algorithm 1.

Algorithm 1 A resource prediction algorithm based on LTR

```
Require: Dataset $S(X, Y)$, Initialize sample pair weight distribution $D$
Ensure: Final Sort Values $H(x)$
1: Initialization $D_{r=1} = D$ by Eq. (2)
2: for $r = 1$ to $T$ do
3:     Using $D$ to train a weak learner
4:     Find weak learner $h_r$ that maximizes $r(h)$ in Eq.(4)
5:     Choose $\alpha_r = 0.5\ln(1 + r(h)_{max})/(1 - r(h)_{max})$
6:     Choose $Rloss$ to make $\sum_{S_j,\ S_k} D_r\left(S_j\left(x_j, y_j\right),\ S_k\left(x_k, y_k\right)\right) = 1$
7:     Update $D_{r+1} = D_r\left(S_j\left(x_j, y_j\right),\ S_k\left(x_k, y_k\right)\right) \times exp(\alpha_r(h_r(S_j\left(x_j, y_j\right) - h_r(S_k\left(x_k, y_k\right))/Rloss$
8:     Calculate $H(x) = \sum_{r=1}^{R} \alpha_r hr$
9: return $H(x)$
10: end for
```

In this algorithm, $S(X, Y)$ denotes the input simulation application features and label data and D denotes the initialized sample pair weight distribution. The Final Sort Values $H(x)$ are calculated by four steps.

(1) Initialize $D_1 = D$ and ensure that the initial D value is the same for each sample pair by setting the value of c in Eq. (2).
(2) During the iterative process, the maximized utility function $r(h)$ is used as the learning objective, weak learning iterative training with D_r is performed to generate multiple weak learners h_r, and the α_r value is obtained by calculation. Finally, the ranking result is calculated as

follows: $H(x) = \sum_{r=1}^{R} \alpha_r \times h_r$. where R denotes the number of iterations and H(x) denotes the ranking score, the higher score indicates the better ranking.

(3) The D distribution value is updated according to the line 7, where $Rloss$ denotes the minimum objective function value in the current iteration calculated using Eq. (5).

(4) Steps 2 and Steps 3 are repeated until the combined ranking order obtained in two to three consecutive iterations does not change, which is then considered the final predicted ranking, and the optimal order of resource usage is the output.

The time-sensitive LTR method for resource prediction

The simulation application resource prediction problem requires a problem to be considered to modify the LTR algorithm. Assume that the optimal resource ranking required for one simulation run is the correct ranking = {4, 2, 3, 1} cores. There are two predicted ranking results for Ranking a = {2, 4, 3, 1} cores and Ranking b = {4, 2, 1, 3} cores. Rankings a and b are incorrectly ranked. However, in simulation practice, one is more concerned with the number of resources that are used to complete the simulation in the shortest time, that is, the number of resources ranked first. Therefore, the time cost of Ranking a is significantly higher than that of Ranking b, where the shortest simulation run time is guaranteed in accordance with the ranking outcome. In other words, ranking a needs to be allotted a larger penalty. However, the RankBoost LTR method does not consider this and assumes that the time cost of Rankings a and b is equal. Therefore, a penalty factor $\varepsilon(x_{j,k})$ is added to ensure that more penalties are added when the best resource is ranked incorrectly. Therefore, the loss function in Eq. (5) can be converted into the following equation:

$$\text{Rloss} = \sum_{S_j, s_k} \varepsilon\left(x_{j,k}\right) \times D_r\left(S_j\left(x_j, y_j\right), S_k\left(x_k, y_k\right)\right) \times \exp\left(\alpha_r\left(h_r\left(S_j\left(x_j, y_j\right) - h_r S_k\left(x_k, y_k\right)\right)\right)\right). \quad (7)$$

The definition of the penalty factor is the main issue with time-sensitive LTR. In this study, we used heuristics to calculate the value of $\varepsilon(x_{j,k})$, which is executed in Algorithm 2.

First, the algorithm establishes a correct ranking based on all computational resource requirements in the group of simulation applications and uses mean reciprocal rank (Eq. 8) as a criterion for ranking performance and for calculating the optimal score MRR_{best}.

Second, for each sample pair $Q = S_j(x_j, y_j), S_k(x_k, y_k)$ in the simulation application group, the positions of their module pairs are exchanged and the new ranking scores $MRR_{j,k}$ are calculated.

Algorithm 2 The calculation of penalty factor

```
Require: Dataset $S(X, Y)$
Ensure: penalty factor $\varepsilon(x_{j,k})$
1: Obtain a correct ranking for all sample pairs in $S(X, Y)$
2: Calculate the $MRR_{best}$ value of the correct ranking
3: Iteration = 0
4: Dec = 0
5: for each pair $Q = S_j(x_j, y_j), S_k(x_k, y_k)$ do
6:     Exchange $S_j\left(x_j, y_j\right)$ and $S_k(x_k, y_k)$
7:     Calculate the MRR value of the new ranking $MRR_{j,k}$
8:     $Dec = Dec + MRR_{best}/MRR_{j,k}$
9:     Iteration++
10: end for
11: $\varepsilon(x_{j,k}) = Dec/Iteration$
12: return $\varepsilon(x_{j,k})$
```

Finally, the standard decline ratio of the MRR values of all sample pairs after the swap relative to the correctly ranked MRR values was calculated and used as the value of the penalty factor $\varepsilon(x_{j,k})$.

Evaluation metrics

To test the accuracy of the ranked learning prediction method, two metrics were considered: mean reciprocal rank (MRR) and normalized discounted cumulative gain (NDCG). The indicator metrics are formally defined as follows:

(1) MRR is concerned with the position of the first relevant element in the sorting result. If the first relevant element appears in the first position, the MRR will be larger. The MRR is calculated as follows:

$$MRR = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{rank_i}, \quad (8)$$

where K denotes the number of samples and rank$_i$ denotes the position where the best resource appears in the predicted computed resource ranking list for the ith sample.

(2) By introducing a location influence factor, $NDCG$ considers that more advanced samples in the ranking are more valuable and have a significant impact on the assessment results. The $NDCG@n$ is calculated as follows:

$$NDCG@n = Z_n \sum_{i=1}^{n} \frac{2^{c(i)} - 1}{log(1+i)}, \quad (9)$$

where Z_n denotes the generalization factor such that the range of NDCG values is limited to [0, 1], $c(i)$ denotes the relevance rank of the sample ranked at position i, $2^{c(i)} - 1$ denotes the evaluation gain value of that sample, and $log(1+i)$ denotes the discount weight of the ranking position of the document, with a smaller value of i indicating a higher sorted position. Finally, $NDCG@n$ accumulates all evaluation gain values of the top n documents in the ranking as the performance evaluation value of the ranking model.

Experimental results and analysis

Application and experimental settings

Phold is a benchmark program used to evaluate the performance of simulation applications [37]. The Phold includes many simulation entities, and each entity will initialize multiple simulation events. When running the Phold, the simulation entities randomly executes three types of events and sends messages to another objects.

Public Opinion Dissemination Model (PODM) is a typical complex system simulation model, that studies the impact on public opinion dissemination when hot events occur. The PODM consist of four types simulation entities: individuals, hot events, cities, and medias.

In this study, we repeat running simulation application with different parameters and collect feature data every 3 s to train multiple predictors. As shown in Table 2, the pre-execution parameters of the simulation application were specified to run in a real cloud environment. To ensure the effectiveness of the algorithm, we repeated each experiments 10 times, and calculated the average value as the experimental result. Then, we built a cloud environment to execute complex simulation application and collect the feature data. The proposed approach was tested on Intel Xeon Gold 6338 CPU with 32 cores and 64 GB of RAM. In this study, the simulation application was built by Docker, an open container engine, and the simulation tasks were offloaded to no more than 32 containers (each container was assigned 1 core and 2 GB RAM) for parallel execution.

Table 2 Pre-execution parameters configuration of Phold and PODM

Simulation application	Features	Values
Phold	CPU cores	[1, 2, 4, 8, 16, 24, 32]
	Entity	[1000, 2000, 3000, 4000, 5000]
	Event	[100, 200, 300, 400, 500]
	Simulation time	[2000]
	Lookahead	[0.2, 0.4, 0.6, 0.8. 1.0]
PODM	CPU cores	[1, 2, 4, 8, 16, 24, 32]
	Individuals	[500, 1000, 1500, 2000]
	Hot events	[5,10,20]
	Cities	[10,20,30]
	Media	[10]
	Simulation time	[1000]
	Lookahead	[0.1, 0.5, 1.0]

Feature extraction method based on the SHAP interpretable framework

The static features of the simulation application and the dynamic features at runtime are among the factors that affect the operational efficiency of complex system simulation applications in the cloud computing environment. Most current feature importance calculation methods can only indicate which features are important without indicating how these features affect the prediction results. Inspired by cooperative game theory, SHAP develops a method to explain individual predictions based on the game theory [38]. Therefore, to extract the significant factors affecting the operational efficiency of simulation applications, this study analyzed the degree of influence of each factor on the operational efficiency of simulation applications based on SHAP's interpretable feature extraction method.

Figure 2 shows an overall feature summary analysis based on SHAP interpretability model. Each point shown in the figure represents the SHAP values of the sample points corresponding to all features. The colors of the points, which range from blue to red, represent the low to high values of each feature. The x-axis in the figure represents the SHAP values; when the SHAP values of the features are greater than 0, it indicates a positive impact on the output of the model; when they are less than 0, it has a negative impact on the output of the model. According to Fig. 2, the feature "Entity" has the largest value, and the feature value of samples is greater than 0. This indicates that the feature has the largest positive impact on the model's output, that is, the simulation runtime increases as the number of simulation entities increases. The feature "File" has the smallest value, which indicates it is the least important to the model and has the smallest impact on its output.

To further investigate the influence pattern of each factor on the simulation runtime, SHAP dependency diagrams of the top three characteristic variables that significantly affect the simulation runtime are drawn, as shown in Fig. 3. The contribution of the number of simulation entities "Entity" has an overall positive correlation, that is, the simulation computational loads and communication loads increases as the increase of the number of simulation entities and cause a positive impact on the simulation runtime. The contributions of the number of CPU cores and CPU usage were negatively correlated over a range, which means that predicting and allocating optimal computing resources to simulation applications can reduce simulation runtime.

To prove the significance of SHAP method, mutual information methods are used to compare the performance of

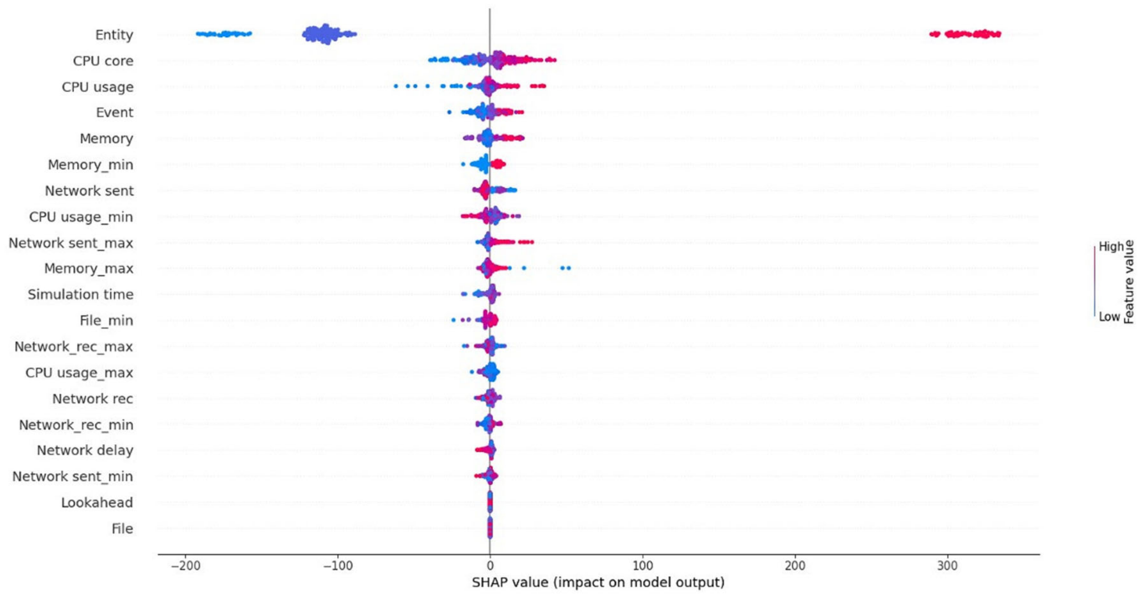
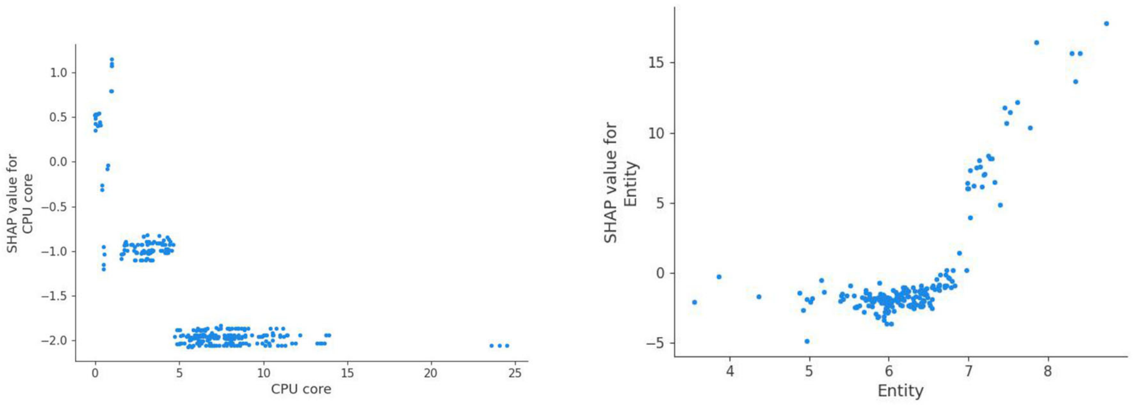
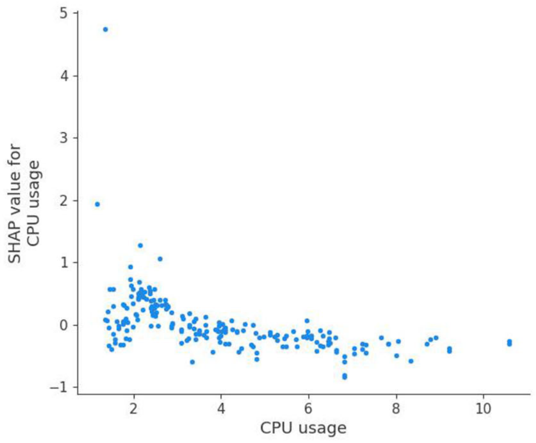


Fig. 2 Overall feature summary analysis based on the SHAP interpretable framework



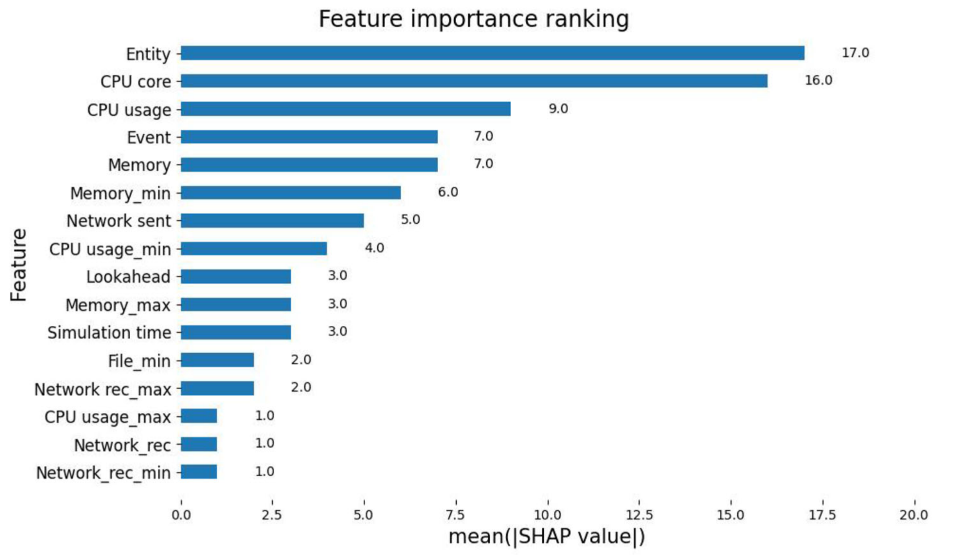
(a) Effect of CPU core on model output (b) Effect of Entity on model output



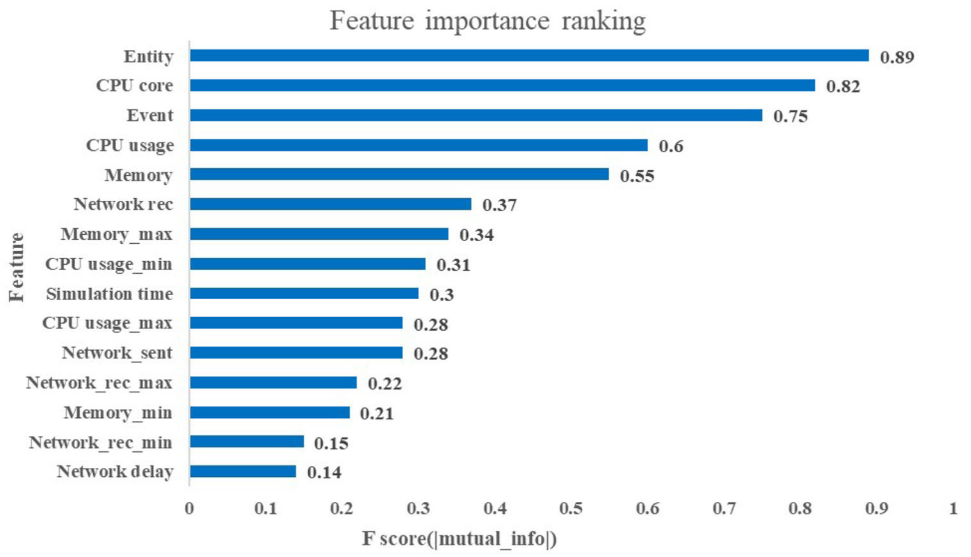
(c) Effect of CPU usage on model output

Fig. 3 Impact of simulation application characteristics on model output

Fig. 4 The effect of each feature on the model performance



(a) The feature importance ranking based on SHAP value



(b) The feature importance ranking based on mutual information

feature selection. First, all features are ranked based on their importance in Fig. 4. In addition, the features that SHAP values less than 1 and the F score less than 0.1 are removed. Second, this study uses the top k feature dimensions in the importance ranking to train the LTR model and evaluates its performance based on the NDCG model performance evaluation metrics. As shown in Fig. 5, The model performance obtained by SHAP method is better than the mutual information method in almost all dimensions, and the highest performance was obtained using the SHAP method with top 16 feature dimensions for training the ranking learning model. Therefore, these features were selected to train the ranking learning model in the subsequent experiments in this study.

Model parameter setting and sensitivity analysis

In this study, eight prediction models such as linear regression (LR), extremely randomized trees (ETR), support vector regression (SVR), extreme gradient boosting (XGBoost), multilayer perceptron (MLP), restricted Boltzmann machine (RBM), ranking boost(Rankboost) and random forest (RF) are built to compare the ranking performance, and the parameters of each models were tuned by the grid search toolkit. The specific parameters are shown in Table 3.

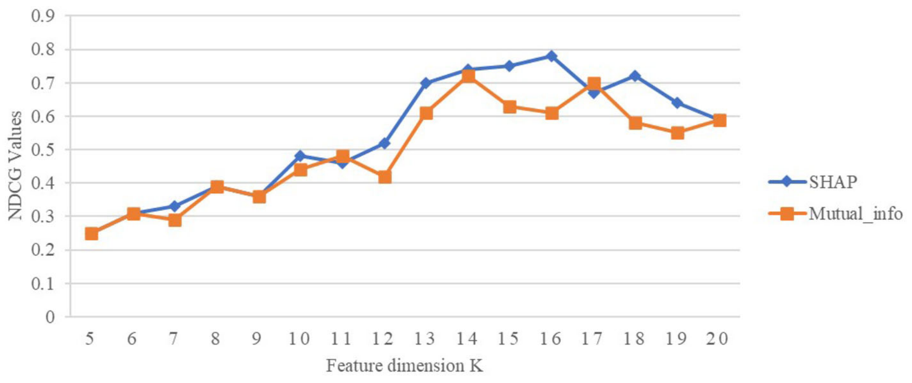


Fig. 5 Performance analysis with different feature dimensions

Table 3 Prediction model and parameter settings

Model	Parameters
ETR	Num of trees = 200
LR	Max iterations = 100, regularization parameter = 0.2
RF	Max depth = 4, num of trees = 300
XGB	Learning rate = 0.05, max depth = 3, Max iterations = 300
MLP	Hidden layer = (200,80), learning rate = 0.1, max iterations = 600
SVR	Kernel = "linear", regularization parameter = 0.1, max iterations = 500
Rankboost	Learning rate = 0.15, max depth = 5, max iterations = 800
RBM	Learning rate = 0.2, max iterations = 400

In addition, we analyze the parameters sensitivity of TSLTR on Phold application in Fig. 6. The model performance can improve as learning rate increases and get the best result when learning rate = 0.08, as shown in Fig. 6(a). When setting different max depth and n_estimators, NDCG values with max depth = 6 and n_estimators = 600 are the best, as shown in Fig. 6(b). In subsequent experiments, the hyper-parameter learning rate is set to 0.08, and max depth and n_estimators are set to 6 and 600, respectively.

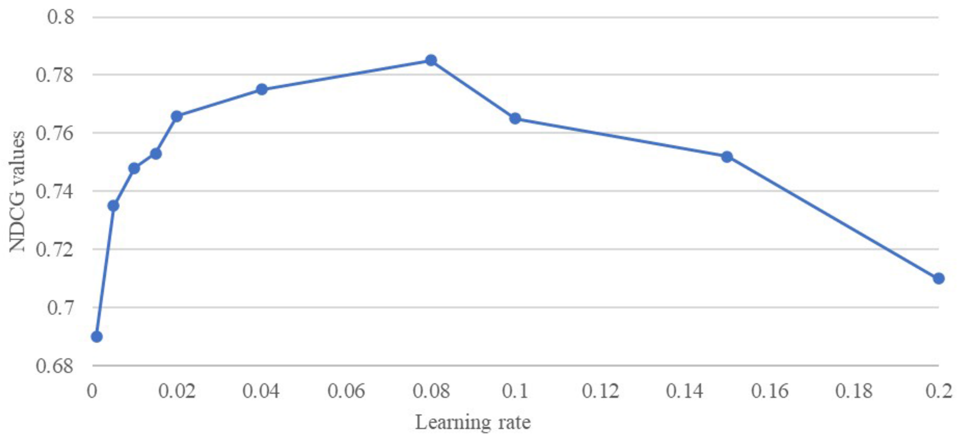
Model performance evaluation

This section evaluates the performance of the time-sensitive LTR approach for resource prediction of cloud simulation (TSLRT). Figures 7 and 8 present the detailed NDCG values and MRR of the nine methods. The TSLRL model achieves the highest NDCG and MRR values on Phold and PODM applications, which the NDCG and MRR values in Phold is 0.78 and 0.81 and in PODM is 0.75 and 0.83. Compared to the other 8 models, The TSLRT improves the average NDCG values of RBM by 45%, of SVR by 40%, of MLP by 38%, of LR by 34%, of ETR by 31%, of XGB by 20%, of RF by 17%, and of Rankboost by 7%, respectively. This is because the TSLRT model modifies the loss function by considering the simulation features, and uses the relative ranking between computing resources as feedback information to train a predictor for minimizing the error ranking probability. Compared to RankBoost and TSLRT, other methods tend to get worse performance in NDCG and MRR values, because their aim to minimize the error between the predicted value and the true value and does not account for the impact of computing resource ranking. After revising the loss function of RankBoost, the TSLRT gains 3%–8% higher average NDCG and MRR values, because the TSLRT takes into account the ranking cost issue and modifying the loss function to produce the best ranking possible for the optimal resource. In addition, different models have their own advantages for different indicators. For example, The NDCG value of MLP model is lower than that of LR, ETR models in Phold application; However, the MRR value of MLP model is higher than that of LR, ETR models. The RBM model gains the lowest NDCG values in Phold but the SVR model has the worst performance in PODM application. According to the experimental results in Figs. 7 and 8, We conclude that the TSLRT achieved a higher-ranking performance in term of NDCG and MRR.

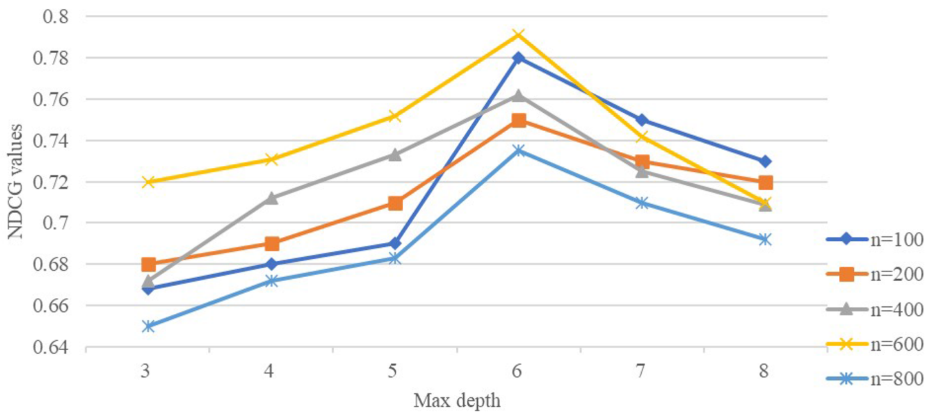
Run-time comparison

Table 4 shows the average training and testing times for all the sorting. Compared with the other methods, TSLRT requires the longest training time of 116.04 s. Additionally, the training times for Rankboost, XBG, MLP, and RBM were 86.683, 30.267, 55.101, and 60.325 s, respectively. This is because these methods also require several iterations to obtain the optimal training model. However, the TSLRT and Rankboost models, with prediction times of 0.55 s and 0.52 s, respectively, produced the quickest results. This is because the other models require multiple prediction runtimes and generate the results by sorting, thus incurring more overhead. In this study, we focused on the prediction of simulation runtime resources under static conditions and trained all prediction

Fig. 6 Parameter sensitivity results

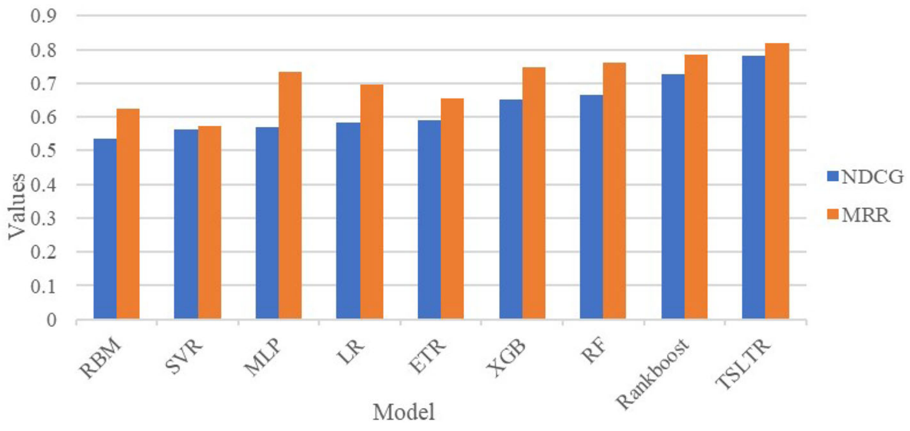


(a) Different learning rate with max depth = 6 and n_estimators = 600



(b) Different max depth and n_estimators with fixed learning rate = 0.08

Fig. 7 Performance evaluation of the different method in Phold application



models offline. Thus, the TSLRT trades off some of the training times to achieve higher prediction accuracy.

The Phold application was run in a cloud environment using two simulation applications with different parameters (the number of simulation entities was 3000 and 100) and executed in parallel using various CPU core counts. In the experiments, the proposed TSLRT method was used to predict the priority of the computational resources used during this run by collecting the feature data of the simulation application after running for 60 s as input and ranking them from the highest to lowest priority. As shown in Table 5, for the simulation application Phold-1, the TSLRT model can accurately predict the priority order of this application using different CPU core counts. For the simulation application Phold-2, the TSLRT model incorrectly predicts only the priority order with 24 and 12 CPU cores. Additionally,

Fig. 8 Performance evaluation of the different method in PODM application

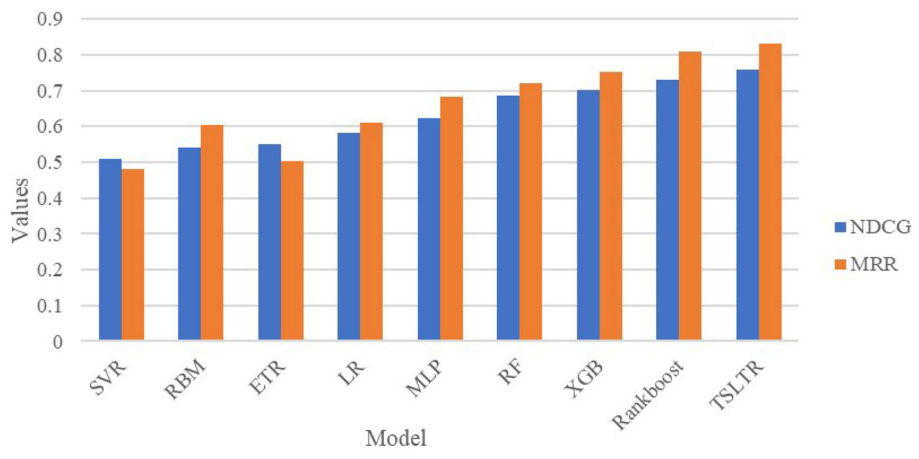


Table 4 Average training and the result output time times of different models

Model	Training time	Ranking result output time
RBM	60.325	3.15
SVR	1.165	1.44
MLP	55.101	2.52
LR	1.035	1.74
ETR	2.012	1.23
RF	2.023	2.58
XBG	30.267	3.25
Rankboost	86.683	0.52
TSLTR	116.04	0.55

Table 5 Comparison of the predicted priority order with the actual order

Simulation application	CPU core usage (Listed in descending order of prediction results)	Actual runtime (s)
Phold_1 (simulation entities are set 3000)	24	456
	32	910
	16	1075
	8	1845
	4	2628
	2	3428
	1	6294
Phold_2 (simulation entities are set 1000)	16	928
	24	1966
	32	1083
	8	3045
	4	5742
	2	10,578
	1	20,253

the shortest simulation runtime was obtained when the highest priority number of CPU cores was used in both cases. Therefore, the experimental results show that the ranking learning method proposed in this study can effectively predict the computational resource priority of the simulation application, and by selecting the highest priority computational resource, the simulation application runtime can be significantly reduced and its efficiency can be improved.

Conclusion

This study proposes a method for predicting cloud simulation computing resource, which is characterized by predicting the overall ranking of computing resource usage during a simulation application operation and obtaining the shortest runtime when the simulation application is run with the computing resource with the highest predicted ranking. First, the simulation application is deployed in the cloud environment to generate the dataset. Second, the SHAP interpretable feature extraction method is used to quantitatively determine the importance of each feature dimension. Third, the feature set that significantly applies the simulation application runtime is evaluated. Finally, this study proposes a cloud simulation computing resource prediction algorithm based on time-sensitive ranking learning, which features the factors that significantly affect the simulation application runtime obtained from the analysis, and constructs a simulation runtime sensitive ranking learning model based on RankBoost by designing the objective loss function in the ranking learning algorithm. Different machine-learning methods were evaluated in this study to demonstrate the advantages of the proposed method. The results demonstrate that the proposed method can effectively predict the overall ranking of simulation computational resources and accurately evaluate the computational resources required for the shortest runtime of a simulation application. In future research, we plan to consider scalability factors to predict the overall ranking of computational resources in different types of simulation applications and to evaluate the model performance on larger scale cloud platforms.

Author Contributions We declare that this manuscript entitled "A time-sensitive learning-to-rank approach for cloud simulation resource prediction" is original, has not been published before and is not currently being considered for publication elsewhere we confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

Data availability The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest we declare that we have no known competing financial interests or personal relationships or organizations that could have appeared to influence the work reported in this paper.

References

1. Yao Y-P, Meng D, Zhu F, Yan L-B, Qu Q-J, Lin Z-W, Ma H-B (2017) Three-level-parallelization support framework for large-scale analytic simulation. J Simul 11(3):194–207. https://doi.org/10.1057/s41273-017-0057-x
2. Fujimoto RM, Malik AW, Park A et al (2010) Parallel and distributed simulation in the cloud. SCS M&S Magaz 3:1–10
3. Fujimoto RM (2016) Research challenges in parallel and distributed simulation. ACM Trans Model Comput Simul. https://doi.org/10.1145/2866577
4. Soni D, Kumar N (2022) Machine learning techniques in emerging cloud computing integrated paradigms: A survey and taxonomy. J Network Comput Appl 205:103419. https://doi.org/10.1016/j.jnca.2022.103419
5. Xiao Y, Yao Y, Chen K, Tang W, Zhu F (2022) A simulation task partition method based on cloud computing resource prediction using ensemble learning. Simul Model Pract Theory 119:102595. https://doi.org/10.1016/j.simpat.2022.102595
6. Kim IK, Wang W, Qi Y, Humphrey M (2022) Forecasting cloud application workloads with cloudinsight for predictive resource management. IEEE Trans Cloud Comput 10(3):1848–1863. https://doi.org/10.1109/TCC.2020.2998017
7. Chen Z, Zhu Y, Di Y, Feng S (2015) Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network. Intell Neurosci 2015. https://doi.org/10.1155/2015/919805
8. Kumar J, Singh AK (2020) Adaptive learning based prediction framework for cloud datacenter networks' workload anticipation. J Inform Sci Eng 36(5):981–992. https://doi.org/10.6688/JISE.202009_36(5).0003
9. Kholidy HA (2020) An intelligent swarm based prediction approach for predicting cloud computing user resource needs. Comput Commun 151:133–144. https://doi.org/10.1016/j.comcom.2019.12.028
10. Zhuang P, Wan Y, Qiao Y (2020) Learning attentive pairwise interaction for fine-grained classification. AAAI Confer Artif Intell 34:13130–13137
11. Vanmechelen K, De Munck S, Broeckhove J (2013) Conservative distributed discrete-event simulation on the amazon ec2 cloud: An evaluation of time synchronization protocol performance and cost efficiency. Simul Model Pract Theory 34:126–143. https://doi.org/10.1016/j.simpat.2013.02.002
12. Seneviratne S, Levy DC (2011) Task profiling model for load profile prediction. Future Gener Comput Syst Int J Sci 27(3):245–255. https://doi.org/10.1016/j.future.2010.09.004
13. Lee B, Schopf J (2003) Run-time prediction of parallel applications on shared environments. In: IEEE International Conference on Cluster Computing, proceedings, pp. 487–491
14. Angiulli F, Fassetti F (2013) Nearest neighbor-based classification of uncertain data. ACM TRANS Knowledge Discovery Data. https://doi.org/10.1145/2435209.2435210
15. Miu T, Missier P (2012) Predicting the execution time of workflow activities based on their input features, pp. 64–72
16. Matsunaga A, Fortes JA (2010) On the use of machine learning to predict the time and resources consumed by applications. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 495–504
17. Shyam GK, Manvi SS (2016) Virtual resource prediction in cloud environment: a bayesian approach. J Network Comput Appl 65:144–154. https://doi.org/10.1016/j.jnca.2016.03.002
18. Ullah A, Nawi NM (2021) An improved in tasks allocation system for virtual machines in cloud computing using hbac algorithm. J Ambient Intell Hum Comput. https://doi.org/10.1007/s12652-021-03496-z
19. Malik S, Tahir M, Sardaraz M, Alourani A (2022) A resource utilization prediction model for cloud data centers using evolutionary algorithms and machine learning techniques. Appl Sci-Basel. https://doi.org/10.3390/app12042160
20. Valarmathi K, Raja SKS (2021) Resource utilization prediction technique in cloud using knowledge based ensemble random forest with lstm model. Concurr Eng-Res Appl 29(4):396–404. https://doi.org/10.1177/1063293X211032622
21. Wang S, Zhu F, Yao Y, Tang W, Xiao Y, Xiong S (2021) A computing resources prediction approach based on ensemble learning for complex system simulation in cloud environment. Simul Model Pract Theory. https://doi.org/10.1016/j.simpat.2020.102202
22. Xia B, Li T, Zhou Q, Li Q, Zhang H (2021) An effective classification-based framework for predicting cloud capacity demand in cloud services. IEEE Trans Serv Comput 14(4):944–956. https://doi.org/10.1109/TSC.2018.2804916
23. Nawrocki P, Osypanka P (2021) Cloud resource demand prediction using machine learning in the context of qos parameters. J Grid Comput. https://doi.org/10.1007/s10723-021-09561-3

24. Shen H, Chen L (2022) A resource-efficient predictive resource provisioning system in cloud systems. IEEE Trans Paral Distrib Syst 33(12):3886–3900. https://doi.org/10.1109/TPDS.2022.3172493
25. Al-Asaly MS, Bencherif MA, Alsanad A, Hassan MM (2022) A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment. Neural Comput Appl 34(13):10211–10228. https://doi.org/10.1007/s00521-021-06665-5
26. Bi J, Li S, Yuan H, Zhou M (2021) Integrated deep learning method for workload and resource prediction cloud systems. NEurocomputing 424:35–48. https://doi.org/10.1016/j.neucom.2020.11.011
27. Yu X, Liu J, Keung JW, Li Q, Bennin KE, Xu Z, Wang J, Cui X (2020) Improving ranking-oriented defect prediction using a cost-sensitive ranking svm. IEEE Trans Reliab 69(1):139–153. https://doi.org/10.1109/TR.2019.2931559
28. Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 133–142
29. Freund Y, Iyer R, Schapire R, Singer Y (2004) An efficient boosting algorithm for combining preferences. J Machin Learn Res 4(6):933–969. https://doi.org/10.1162/1532443041827916
30. Rudin C, Schapire RE (2009) Margin-based ranking and an equivalence between adaboost and rankboost. J Machin Learn Res 10:2193–2232
31. Ganjisaffar Y, Caruana R, Lopes CV (2011) Bagging gradient-boosted trees for high precision, low variance ranking models. In: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 85–94
32. Nguyen TT, An TQ, Hai VT, Phuong TM (2014) Similarity-based and rank-based defect prediction. In: 2014 International Conference on Advanced Technologies for Communications (ATC 2014), pp. 321–32
33. Yang X, Tang K, Yao X (2015) A learning-to-rank approach to software defect prediction. IEEE Trans Reliabil 64(1):234–246. https://doi.org/10.1109/TR.2014.2370891
34. Burges C, Svore K, Bennett P, Pastusiak A, Wu Q (2011) Learning to rank using an ensemble of lambda-gradient models. In: Proceedings of the Learning to Rank Challenge, pp. 25–35
35. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139. https://doi.org/10.1006/jcss.1997.1504
36. Karatza HD, Stavrinides GL (2019) Modeling and simulation of cloud computing and big data. Simul Model Pract Theory 93:1–2. https://doi.org/10.1016/j.simpat.2019.01.003
37. Yoginath SB, Perumalla KS (2015) Efficient parallel discrete event simulation on cloud/virtual machine platforms. ACM Trans Model Comput Simul. https://doi.org/10.1145/2746232
38. Baptista ML, Goebel K, Henriques EMP (2022) Relation between prognostics predictor evaluation metrics and local interpretability shap values. Artif Intel 306:103667. https://doi.org/10.1016/j.artint.2022.103667