ORHRC: Optimized Recommendations of Heterogeneous Resource Configurations in Cloud-Fog Orchestrated Computing Environments

Ai Xiao[†§], Zhihui Lu[*†‡], Xin Du[†§], Jie Wu[*†¶], and Patrick C.K. Hung[‖]
[†]*School of Computer Science, Fudan University, Shanghai, China*
[§]*Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, Shanghai, China*
[‡]*Shanghai Blockchain Engineering Research Center, Shanghai, China*
[¶]*Peng Cheng Laboratory, Shenzhen, China*
[‖]*Faculty of Business and Information Technology, Ontario Tech University, Canada*
Email: axiao18, lzh, xdu20, jwu@fudan.edu.cn
and patrick.hung@uoit.ca

***Abstract*—The cloud-fog orchestrated computing environments devolve computing tasks from the cloud center to the fog nodes, providing more heterogeneous configurations for the operation of workloads. Compared to the conventional cloud computing environment, the physical conditions at the fog nodes in the cloud-fog orchestrated computing environments are more complex and changeable. Therefore, the configurations that the fog nodes provide are heterogeneous and varying. This requires the configuration selection model to adapt to changeable configurations. The previous configuration selection models are applied to the limited and fixed configurations in the conventional cloud environment, but not to the complex cloud-fog orchestrated computing environments. To address this problem, we propose Optimized Recommendations of Heterogeneous Resource Configurations(ORHRC), a model that provides users with a reliable cloud configuration recommendation service. ORHRC uses the matrix factorization algorithm and neural network to build a recommendation model, which combines the operating characteristics of workloads as the explicit ratings and implicit feedback, to give configuration recommendations. Comprehensive experiments on a real-world dataset demonstrate that the hit rate of configurations of ORHRC is 24% higher than Micky and 15% higher than Selecta.**

***Keywords*-cloud-fog orchestrated computing; cloud configuration; recommendation system; resource allocation; heterogeneous resource**

I. Introduction

With the rapid development of cloud computing technology, the public cloud platform can stably and comprehensively provide virtual resource support for data processing tasks of users [1]. Cloud platform service providers(e.g., Amazon) integrate the resources in the cloud into virtual resource pools and form a variety of instance types for users to choose to process their workloads. The instance types mainly include the number of CPUs, memory size and storage size. Besides, they also include configuration items such as network bandwidth and the number of network cards.

Although a conventional centralized cloud data center has a huge pool of computing and storage resources, it is also difficult to satisfy growing data processing requests of users [2]. The emergence of fog computing has alleviated the problems of large bandwidth consumption, communication delay, and IO bottlenecks in cloud data centers [3]. Compared with cloud computing, fog computing mainly relies on distributed computing resources that are close to edge terminal devices rather than remote servers located in the cloud center. Based on the geographical characteristics of fog nodes and the different resource requirements of users in various fields, the internal structure of the server clusters in fog nodes is complicated. At the same time, the fog nodes are more variable than the cloud center, including changes in internal architecture and updates of physical hardware. These characteristics lead to the heterogeneity and variability of the configurations provided by the fog nodes. The more heterogeneous edge terminal devices(e.g., user mobile devices) in the edge nodes [4] are temporarily ignored. Because the resources of each device are small and it is difficult for the cloud platform to identify and control devices. In general, cloud computing and fog computing are integrated, and we call it cloud-fog orchestrated computing.

The service providers in the cloud-fog orchestrated computing platform provide a variety of configurations based on the different requirements of users [5]. For example, AWS EC2 [6] divides the configurations into 6 different groups and the number of these configurations in groups ranges from 1 to 40. The configurations are updating constantly and iteratively. Users can choose the configurations for workloads according to their requirements. However, it has a high probability to bring huge additional loss if users choose configurations according to personal intuitive. Therefore, the cloud-fog orchestrated computing platform needs to provide users with reliable configuration recommendations.

Besides, bandwidth consumption caused by data transmis-

Corresponding authors: Zhihui Lu and Jie Wu

978-1-7281-8786-0/20/$31.00 ©2020 IEEE
DOI 10.1109/ICWS49710.2020.00059

sion in the conventional cloud environment is also a problem. The previous researches are based on the conventional cloud environment. The input of models designed by these researches includes workloads uploaded by users. So, the users need to upload their workload to a geographically distant cloud center. As the scale of workloads increases, the process of uploading workloads to the cloud center takes up a large amount of network bandwidth and increases the communication latencies [7].

In order to address the above-mentioned challenges, we designed ORHRC, a model that provides users with reliable configuration recommendation services in complex heterogeneous cloud-fog orchestrated computing environments. ORHRC uses matrix factorization algorithms with a neural network to build models. The model combines explicit ratings and implicit feedback based on the operating characteristics of workloads to provide configuration recommendations. In the period of model training, we first collect the operating characteristics of each training workload on various configurations. These operating characteristics are used to characterize the workload. Secondly, we convert the operating characteristics of the workload into the implicit feedback and convert the execution time of the workload into the explicit ratings. The implicit feedback and explicit ratings of each workload will be formed as the data matrix of the ORHRC model. Finally, the ORHRC model can be trained according to the data matrix of training workloads and predict the ranking of the execution time of all workloads under different configurations. The first and second steps are processed by the Configuration Selection module of ORHRC. The final step is processed by the Performance Prediction module of ORHRC. The two modules will be described in Section III-C.

As shown in Figure 1, we put the Performance Prediction module of ORHRC in the cloud layer and put the Configuration Selection module of ORHRC in the nodes of Fog layer. ORHRC processes workloads in the fog nodes and sends the operating characteristics of workloads to the cloud layer for modeling. Therefore, most of the data that users upload to the cloud layer is transferred to the fog nodes, which greatly reduces the occupation of network bandwidth. In the first step, users upload workload to nearby fog node. Secondly, the ORHRC model randomly selects several fog servers to collect the operating characteristics of the workload, and uses these operating characteristics to characterize the workload. In the third step, the fog node uploads these operating characteristics to the ORHRC model in the cloud layer. This greatly reduces the amount of data sent to the cloud and reduces the consumption of network bandwidth. In the fourth step, after the ORHRC model finds the optimal configuration for the workload, it sends a message to the uploading fog node. This message includes detailed information about the optimal configuration of the workload and a list of servers recommended for placement. Our research is to focus on the

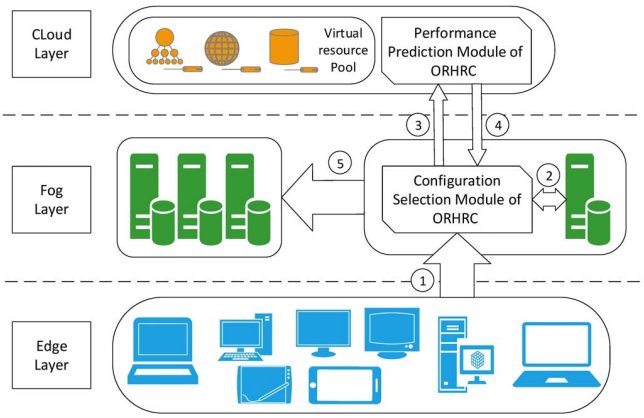

Figure 1: Data Transmission in Cloud-Fog Orchestrated Computing Environments

recommendation of configuration rather than the selection of server to placement, so we simply select the servers that contain the recommendation configuration and near the uploading fog node as the recommendation servers. In the fifth step, the workload is sent to the server that is recommended from the message for operating.

We evaluate ORHRC with 107 workloads on 18 configurations. Under different evaluation methods, the average prediction accuracy of our ORHRC can achieve 84%, which is higher than the state-of-art methods in 24% and 15%. The experiment results may be biased under other datasets.

The main contributions can be summarized as follows:

1) We propose ORHRC, a recommendation model to recommend configurations for workloads in the complex and changeable cloud-fog orchestrated computing environments.
2) We convert the operating characteristics of workloads to implicit feedback and the execution time of workloads to explicit ratings. The explicit ratings and implicit feedback are combined as the data matrix in our ORHRC model.
3) We combine ORHRC with our previous work-SARA [8] to alleviate the cold start problems. In this way, our experiment result shows that SARA can improve the prediction accuracy of ORHRC by 18%.

The rest of this paper is organized as follows. Section II introduces the related works. Section III presents the details of the ORHRC model. Section IV evaluates the experiment results of ORHRC. The conclusions and future work are summarized in Section V.

II. Related Work

In recent years, many researches have devoted to finding the optimal configuration for user workloads in cloud envi-

ronments. We divide these research results into two types: single-workload models and multi-sample models.

A. Single-workload Models

Ernest [9], CherryPick [10], and Arrow [11] build models for each workload, and all these models have the problems of insufficient flexibility and large model training costs.

Ernest [9] models a single workload and predicts the running state of the workload under different configurations. CherryPick [10] is built for repetitive workloads. Based on the bayesian optimization algorithm, it can quickly find the near-optimal configuration of workloads after multiple runs. Arrow [11] proposes and solves the vulnerability problem in CherryPick. It reduces the searching cost to find the optimal configuration.

B. Multi-sample Models

Many researches build multi-sample models with a group of samples. These models are well universal and reduce the searching cost of finding the optimal configuration.

PARIS [12] considers the tolerance of users for configuration overhead and the acceptance of execution time. It establishes a performance-overhead trade-off function and trains model by the random forest algorithm. Micky [13] uses a multi-armed bandit algorithm to find an optimal configuration for a large number of workloads. It can greatly reduce the searching cost. However, the selected optimal configuration is only suitable for most workloads, and the remaining workloads suffer performance loss. SERAC3 [14] uses neural networks to classify big data analysis workloads. For each group of workloads, it uses the bayesian optimization algorithm to determine the optimal configuration of these workloads. Our previous work-SARA [8] uses deep reinforcement learning to quickly find the near-optimal cloud configuration for big data workloads. To minimize the searching cost, SARA makes compromises in accuracy. Selecta [15] focuses on the selection of storage configurations for data-driven workloads. The types of workloads and configurations used for training are limited. ARVMEC [16] uses an XGBoost-based ensemble learning algorithm to make accurate predictions on workload performance for all VM types according to multiple purposes.

In summary, existing researches are designed for conventional cloud computing environments. The number of configurations in the environment is limited and the types of instances are fixed. It is difficult for these researches to adapt to the diverse and heterogeneous configuration conditions in the cloud-fog orchestrated computing environments. On the contrary, our ORHRC model can adapt to these conditions.

III. Model Design

In this section, we discuss the detail of our ORHRC model. We introduce the problem definition of our research in Section III-A. We explain the definition and conversion methods of implicit feedback and explicit ratings in Section III-B. Then, we describe the model design details in Section III-C. Finally, we introduce how our ORHRC model alleviates the cold start problems in Section III-D.

A. Problem Definition

ORHRC is designed to find optimal configurations for workloads in a complex and changeable cloud-fog orchestrated computing environments. The cloud-fog orchestrated computing environments provide a variety of configurations for users to choose, the space of these configurations is designed as $C = \{c_1, ..., c_n\}$, where c_i is a vector, including configuration settings such as CPU, memory, storage and so on. The workloads of users is defined as $W = \{w_1, ..., w_m\}$. The behavior of the workload is difficult to observe and describe intuitively. Therefore, we collect the operating characteristics of workloads to characterize them. Different workloads have different operating characteristics when running on different configurations. So, we use x_{ij} to describe the operating characteristics of workload w_i on configuration c_j. Table 1 shows the detail of the operating characteristics we collect. At the same time, we use t_{ij} to describe the execution time of workload w_i on the configuration c_j.

$$t_{ij}, x_{ij} = f_1(w_i, c_j) \tag{1}$$

As shown in Formula 1, we collect the operating characteristics and execution time of workload w_i on the configuration c_j at fog nodes. The function f_1 is used to collect this information, which will be explained in detail in section III-B. In general, we need to obtain more data for model training. For each workload w_i, we randomly select some configurations to form configuration set $C_{i*}(C_{i*} \in C)$. After several operations, the X_{i*} set and T_{i*} set are obtained, which contains the operating characteristics and execution time of w_i running on all configurations in the configuration set C_{i*}. Finally, the ORHRC model is trained by the X_{i*} and T_{i*} sets of training workloads. In the prediction period of the ORHRC model, we obtained the X_{k*} and T_{k*} sets of the workload w_k in fog nodes.

$$S_{k,all} = \phi(X_{k*}, f_2(T_{k*})) \tag{2}$$

As shown in Formula 2, $S_{k,all}$ means the execution time ranking of w_k under all configurations, f_2 normalizes the execution time T_{k*} to facilitate ranking prediction. Based on the ranking results, ORHRC can give suitable configuration recommendations for workloads.

B. Implicit Feedback and Explicit Ratings

The data matrix of the ORHRC model is the combination of implicit feedback and explicit ratings. Implicit feedback usually refers to the historical data that does not directly indicate the preferences of users. For example, the behaviors of users such as clicking, browsing, and forwarding can

Table I: Operating Characteristics of Workloads

Category	Characteristics	Explanation
CPU Utilization	cpu.%usr	The percentage of the time CPU spends in user space
	cpu.%sys	The percentage of the time CPU spends in kernel space
Memory Utilization	memory.kbmemfree	The size of free physical memory
	memory.kbmemused	The size of used physical memory
Paging Utilization	paging.pgpgin/s	The size of pages swapped in from disk per second (KB)
	paging.pgpgout/s	The size of the page swapped out to disk per second (KB)
Network Utilization	network.rxkB/s	The data receiving rate
	network.txkB/s	The data transmitting rate

be regarded as the implicit feedback for users to items. In general, the formula of implicit feedback y_{ui} is as follows:

$$y_{ui} = \begin{cases} 1, \ interaction\ (user\ u, item\ i)\ is\ observed \\ 0, \ otherwise \end{cases} \quad (3)$$

As shown in Formula 3 , implicit feedback has no negative feedback. It cannot be determined whether the user prefers the item or not [17]. When y_{ui} is 1, it means that user u and item i have interaction records, but it does not mean the u really likes i. Similarly, a value of 0 does not mean that u does not like i.

The Formula 3 does not fit our configuration recommendation model. The fundamental difference is that in the general recommendation system, the interaction between users and items can imply the evaluation of user for items, although it is not obvious and contains noise. In our model, the user is equivalent to the workload, and the item is equivalent to the configuration. For each workload, we randomly select a certain amount of configurations to collect operating characteristics of workloads, so as to train the model and predict suitable configuration for these workloads. The interaction between workloads and configurations is random, and it cannot imply the evaluation of workloads for configurations. Therefore, we use the operating characteristics of workloads as implicit feedback. The same workload has different operating characteristics on different configurations. These different operating characteristics imply the evaluation of workloads for configurations. As shown in Formula 4, we use x_{ij} and $\vec{0}$ as implicit feedback y_{ij} in our ORHRC model. The criterion is whether the workload w_i operates on the configuration c_j.

$$y_{ij} = \begin{cases} x_{ij}, \ w_i\ operates\ on\ c_j \\ \vec{0}, \ w_i\ does\ not\ operate\ on\ c_j \end{cases} \quad (4)$$

As shown in Table I, we design four categories and select two items of each category to form the operating characteristics of workloads. These operating characteristics can reflect the resource consumption of the workload on different configurations, so as to implicitly feedback the evaluation of workloads for configurations.

Explicit ratings include the behaviors in which users express their preferences for items. In various recommendation systems, they are usually expressed as the ratings of users. In our model, the execution time of workloads is used as the explicit rating. The range of execution time is large, and we need to normalize it. We imitate the scoring mechanism of the conventional recommendation system and map the execution time of workloads to limited discrete scores. It converts the prediction of the execution time into the score of different configurations for workloads. As the goal of our model is to give a suitable configuration recommendation, it does not need to accurately predict the execution time of the workload on all configurations, but only needs to predict the ranking of execution time on these configurations.

C. Model Design

In this section, we will introduce the details of our ORHRC model. As shown in Figure 2, our model is divided into two modules: Configuration Selection module and Performance Prediction module. The Configuration Selection module is responsible for the selection and acquisition of training samples. The Performance Prediction module is responsible for predicting the ranking of execution time for workloads on different configurations.

1) Configuration Selection Module: As mentioned in Section III-A, for each workload w_i in the workloads set W, we need to select some configurations to form a configuration set C_{i*}. And then we use each configuration in C_{i*} to run and collect the operating characteristics of w_i to form the input of the Performance Prediction module. There are two methods to form C_{i*}, one is to select configurations randomly, and the other is to select by SARA [8].

The random selection will bring the user cold start problem, but our ORHRC model can alleviate the user cold start problem with SARA. The SARA model clusters workloads into groups and finds the optimal configuration for each cluster of workloads based on deep reinforcement learning. SARA solves the problem of quickly finding the near-optimal configuration for the incoming workload.

To reduce the measurement cost, SARA makes a compromise in accuracy. The configuration recommended by SARA for workloads can be considered as a sub-optimal

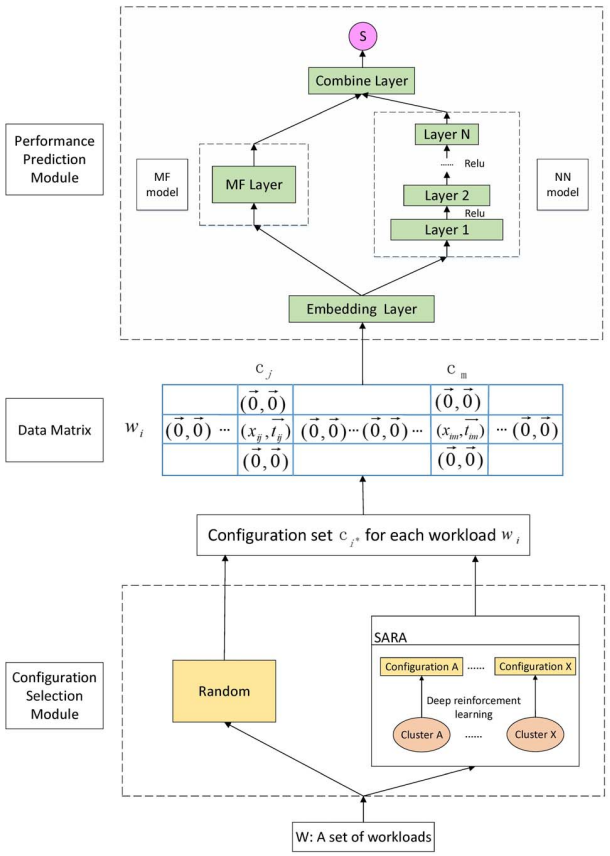

Figure 2: Design of ORHRC Model

configuration. According to the execution time of workload on its sub-optimal configuration, the ORHRC model will provide a more reliable rating for the execution time of workloads on different configurations. Based on the Performance Prediction module, ORHRC can further optimize these sub-optimal configurations. The use of SARA will bring additional measurement cost, but the SARA model is inherently characterized by low measurement cost. So SARA has a limited impact on the measurement cost of our ORHRC model.

Users can also choose whether to use the SARA model according to their needs. Therefore, the Configuration Selection module can build a configuration set C_{i*} for each workload w_i randomly or through the SARA model.

After the configuration construction and workload operation in the fog nodes, we obtained the Data Matrix of the Performance Prediction module. As mentioned in Section III-A, x_{ij} and t_{ij} are the operating characteristics and execution time of workload w_i on configuration c_{i*}. To facilitate the training of the ORHRC model, we expand t_{ij} to $\vec{t_{ij}}$. As shown in Formula 5, $\vec{t_{ij}}$ is a sparse vector, except for the execution time at configuration c_j, the others are 0.

$$\vec{t_{ij}} = (0, 0, ..., f_2(t_{ij}, T_{i*}), 0, ...) \tag{5}$$

T_{i*} is the execution time of workload w_i in all configurations of C_{i*}. The function f_2 is a normalized function. Based on T_{i*}, f_2 maps t_{ij} to a limited number of ratings.

2) Performance Prediction Module: Inspired by the NCF [17] model, we construct the Performance Prediction module of the ORHRC model.

In the Performance Prediction module, we use the Embedding Layer to convert the $\vec{t_{ij}}$ in the Data Matrix into a dense vector for subsequent data processing. The Embedding Layer can convert sparse and large dimensional vectors into dense and low dimensional vectors to visualize the relationship between different discrete variables. Then, we explore the linear and nonlinear relationship between workload and configuration based on the matrix factorization(MF) model and neural network(NN) model. In the Combine Layer, the outputs of the MF model and the NN model are integrated to predict the ranking S for different configurations.

The MF model is a classic recommendation model. The input of the model is the users' rating matrix for items, and the output is the User matrix and the Item matrix. Each row of the User matrix represents a user vector, and each column of the Item matrix represents an item vector. The predicted ratings for users to items are represented by the inner products of user vectors and item vectors. Generally, the User matrix and the Item matrix are trained by minimizing the difference between the predicted ratings and the actual ratings. This minimization process may cause over-fitting problem, so the MF model need to avoid it. In our MF model, we treat workloads as users and configurations as items. As shown in Formula 6, we obtain the latent feature $\hat{y_{ij}}$ of workload vector p_i and configuration vector q_j by the element-wise product method, thus showing the linear preference of workload for configuration. The output of the MF model is a matrix, which is called ϕ_{MF}. Each element of the ϕ_{MF} matrix is the latent feature $\hat{y_{ij}}$. For the over-fitting problem, we use the L2 regularization to solve it. The detail of L2 regularization is shown in Formula 7, The t_{ij} is the execution time, and the λ is a parameter that controls the size of the regularization part. The m is the number of workloads and the n is the number of configurations. We use the Stochastic Gradient Descent(SGD) to iterative the MF model.

$$\hat{y_{ij}} = f_{MF}(i, j \mid p_i, q_j) = p_i^T q_j \tag{6}$$

$$L = \sum_{i=1}^{m}\sum_{j=1}^{n}(t_{ij} - \hat{y_{ij}})^2 + \lambda(\sum_{i=1}^{m} || \, p_i \, ||^2 + \sum_{j=1}^{n} || \, q_j \, ||^2) \tag{7}$$

The MF model and NN model are parallel, they have the same input. In the NN model, the NN framework is used

to fit the nonlinear relationship between workload vector and configuration vector to capture the high-level feature information of them. As shown in Formula 8, The pair of workload vector p_i and configuration vector q_j is the input of the NN model, and l_k is the k-th layer of the model. W_k^T is the weight matrix of l_k, and b_k is the bias term of l_k. Function f is the activation function Relu. Similar to the MF model, the output of NN model is a matrix, which is called ϕ_{NN}. Each element of the matrix is the output of the N-th layer.

$$\begin{aligned} l_1 &= W_1^T(p_i, q_j) \\ l_2 &= f(W_2^T l_1 + b_2) \\ l_k &= f(W_k^T l_{k-1} + b_k) \quad k = 3, ..., N-1 \\ l_N &= f(W_N^T l_{N-1} + b_N) \end{aligned} \tag{8}$$

We connect the output of the MF model ϕ_{MF} and the output of the NN model ϕ_{NN} to form the input of the Combine Layer. Then, the Combine Layer uses a fully connected layer to process the input. As shown in Formula 9, σ is the activation function sigmoid of the Combine Layer. H^T is the edge weights.

$$S = \sigma(H^T(\phi_{MF}, \phi_{NN})) \tag{9}$$

Combine Layer combines the linear and nonlinear characteristics of workload vector and configuration vector, and predict the ranking S of configurations for different workloads. Based on the ranking results, the ORHRC model can recommend suitable configurations for workloads.

D. Cold Start Problems of ORHRC Model

The cold start problem refers to designing a personalized recommendation system without large amount of data and satisfying users with the recommendation results. Cold start problems are generally divided into system cold start problems, user cold start problems and item cold start problems.

The system cold start problems refer to how to give recommendations to new users in an initial system without previous information. In our ORHRC model, we use the big data benchmark workloads as the initial workloads for model training. The model trained by initial workloads can provide suitable configuration recommendations for new users.

The user cold start problems refer to how to give personalized recommendations to new users. In ORHRC, new users refer to incoming workloads. As mentioned in Section III-C, we can integrate the SARA model to select the near-optimal configuration for new workloads. The near-optimal configuration can be regarded as an input to the Performance Prediction module to reduce the performance degradation caused by random selection.

The item cold start problem refers to how to recommend new items to users. This requires real-time integration of new configurations into the model. Based on the initial workloads, the running characteristics and execution time of these workloads on the new configuration can be obtained to form training samples. At the same time, the embedding layer of the ORHRC model can give a general mapping rule, which can map the new configuration to the existing hidden characteristic space, to realize the integration of new configurations.

Therefore, the cold start problems of the ORHRC model can be greatly alleviated.

Table II: Configurations Used in Dataset

Configuration Category	Family	Type
compute optimized	c4	c4.large, c4.xlarge, c4.2xlarge
	c3	c3.large, c3.xlarge, c3.2xlarge
memory optimized	m4	m4.large, m4.xlarge, m4.2xlarge
	m3	m3.large, m3.xlarge, m3.2xlarge
general purpose	r4	r4.large, r4.xlarge, r4.2xlarge
	r3	r3.large, r3.xlarge, r3.2xlarge

IV. Experiment and Results Analysis

In this section, we evaluate the performance of ORHRC based on the dataset collected on AWS EC2. We compare ORHRC with the state-of-art methods Micky and Selecta. Experiments under various evaluation methods show that our ORHRC model has better performance.

A. Experiment Setup

1) Dataset: To evaluate our ORHRC model, we use a public dataset [18] for experimentation.

The dataset obtains 107 real-world workloads from the big data benchmark platforms Hibench [19] and spark-perf [20]. These workloads include various types of workloads, including machine learning, SQL, Micro, Web Search and Graph. The dataset contains 18 configurations on AWS EC2 to operate workloads. As shown in Table II, the dataset mainly selects three categories of configurations from AWS EC2 and chooses six configurations from each category. For each measurement, the dataset collects 72 low-level performance metrics. As shown in Table I, we selected 8 operating characters for the experiment.

2) Evaluation Methods: Based on the leave-one-out method, we use two evaluation methods to evaluate the model performance.

We iterate through the workloads in the dataset, select one workload as the test set each time, and use the remaining workloads as the training set. The evaluation method is as follows:

a) Hit rate of optimal configuration and second-optimal configuration

According to the dataset, we can obtain the optimal configuration c_{io} that workload w_i performances best and the second-optimal configuration c_{is} with the second-highest

performance. At the same time, the ORHRC model gives the prediction configuration c_{ip}.

If the prediction configuration c_{ip} is the same as the optimal configuration c_{io}, the hit count h_i of workload w_i is set as 1. Finally, the hit rate of optimal configuration H_o is calculated as Formula 10. The N means the number of workloads for prediction.

$$H_o = \frac{\sum_{i=1}^{N} h_i}{N} * 100\%,\ h_i = \begin{cases} 1,\ if\ c_{ip} == c_{io} \\ 0,\ otherwise \end{cases} \quad (10)$$

Similarly, if the prediction configuration c_{ip} is the same as the second-optimal configuration c_{is}, the hit rate of second-optimal configuration H_s is calculated as Formula 11.

$$H_s = \frac{\sum_{i=1}^{N} h_i}{N} * 100\%,\ h_i = \begin{cases} 1,\ if\ c_{ip} == c_{is} \\ 0,\ otherwise \end{cases} \quad (11)$$

For the convenience of calculation, we sometimes add the hit rate of optimal configuration H_o and the hit rate of second-optimal configuration H_s to form the hit rate of top-optimal configuration H_t.

$$H_t = H_o + H_s \quad (12)$$

b) Hit rate of near-optimal configuration

According to the execution time of workload on all configurations, we regard these configurations whose execution time is within 110% of the shortest time as near-optimal configurations.

As shown in Formula 13, the r_{io} means the execution time of workload w_i on optimal configuration, and the r_{ip} means the execution time of workload w_i on prediction configuration. If the r_{ip} is within 110% of the r_{io}, the prediction configuration is called the near-optimal configuration, and the hit count h_i of workload w_i is set as 1. Finally, the hit rate of near-optimal configuration H_n is calculated as Formula 13. The N also means the number of workloads for prediction.

$$H_n = \frac{\sum_{i=1}^{N} h_i}{N} * 100\%,\ h_i = \begin{cases} 1,\ if\ r_{ip} < r_{io} * 1.1 \\ 0,\ otherwise \end{cases} \quad (13)$$

3) Baseline Methods: We use the following state-of-art methods as baseline-method to compare with our ORHRC model.

· **Selecta [15]**: using latent factor collaborative filtering to predict near-optimal cloud computing and storage resources for data-intensive workloads.

· **Micky [13]**: using the multi-armed bandit algorithm to find the appropriate configuration for collective workloads.

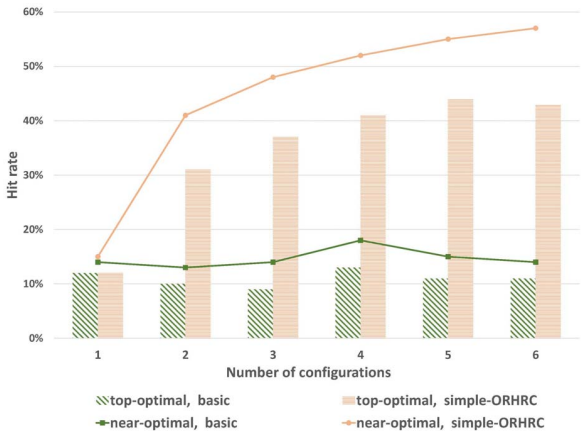

Figure 3: Determining the Number of Testing Configurations

B. Number of Testing Configurations

As mentioned in Section III-B, our ORHRC model needs to select some testing configurations to collect the operating characters of workloads. So we will determine the number of testing configurations based on the experiment result.

In order to exclude the influence of operating characters on the experiment results, we use the id of workload to characterize the workloads, rather than the operating characters of workloads. The id of each workload is a one-hot vector with an activation point of 1, and we convert it into a dense vector through the embedding layer. We call the model with this input as simple-ORHRC.

As shown in Figure 3, we design the basic model that the workload vector and the configuration vector are all one-hot vectors with an activation point of 1 for comparison. With the increase in the number of testing configurations, the hit rate of basic model remains stable, no matter in hitting of top-optimal configurations or near-optimal configurations. For simple-ORHRC model, the hit rate increases with the growth of the number of testing configurations. We find that after 3 testing configurations, the slope of the simple-ORHRC line decreases and the growth trend of the simple-ORHRC bar also slows down. Therefore, we choose 3 testing configurations to collect the operating characters of workloads.

C. Performance of ORHRC

In this section, we discuss the performance of the ORHRC model. We divide the ORHRC model into the random-ORHRC model and SARA-ORHRC model based on randomly selecting configurations or SARA. As shown in Figure 4, under the two evaluation methods, random-ORHRC and SARA-ORHRC have better performance than the state-of-art methods Micky and Selecta. In detail, for the rate of hitting top-optimal configurations and near-optimal configurations, random-ORHRC is 12% and 15% higher

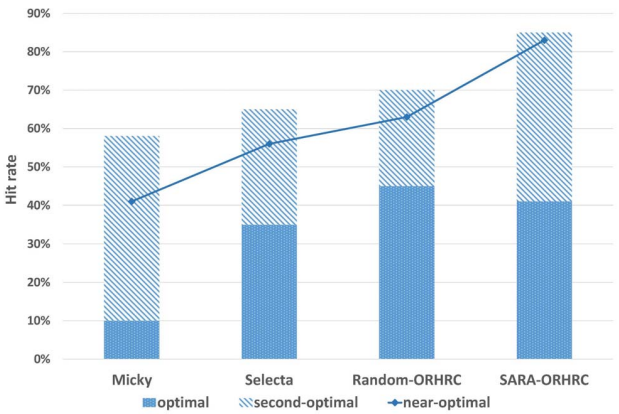

Figure 4: Performance Comparison of Models under Hit Rate of Optimal Configuration, Second-optimal Configuration and Near-optimal Configuration

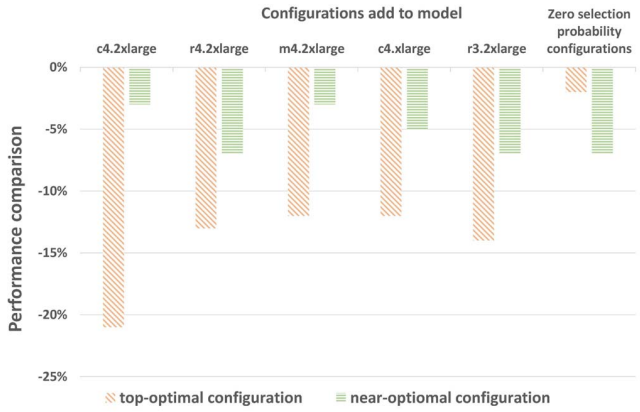

Figure 5: the Performance Loss Caused by the Item Cold Start Problem

than Micky, and 5% and 7% higher than Selecta; SARA-ORHRC is 27% and 42% higher than Micky, and 20% and 27% higher than Selecta. Above all, the average hits rate of ORHRC is 24% higher than Micky and 15% higher than Selecta. Besides, the results show that SARA allows ORHRC to improve the accuracy of predicting second-optimal and near-optimal configurations by average 18%. It can be considered that the addition of SARA has indeed alleviated the user cold start problem of the ORHRC model. Besides, our random-ORHRC and SARA-ORHRC use benchmark workloads rather than actual workloads for the experiment. It also proves that the system cold start problem can be alleviated in our ORHRC model system.

D. Alleviation of the Cold Start Problem

In Section IV-C, we discussed the user cold start problem and the system cold start problem of the ORHRC model. In this section, we will discuss the item cold start problem of ORHRC.

We delete a configuration from the dataset as the new add configuration. Then, we train the ORHRC model based on the remaining configurations. After the model training is completed, we randomly select half of the workloads to run on the deleted configuration and collect data samples to iteratively train the model. Finally, the model predicts the optimal configuration of all workloads.

As shown in Figure 5, we select configurations that are frequently selected as the optimal configurations from the dataset for testing, including c4.2xlarge, r4.2xlarge, m4.2xlarge and so on. We call these configurations as high selection probability configurations.

The high selection probability configurations have a performance loss of more than 10% in the hit rate of top-optimal configurations, especially c3.2xlarge, which has a performance loss of about 20%. This can be explained because the high selection probability configurations are the optimal configurations for plenty of workloads. The lack of training samples about these configurations will seriously affect the hit rate of top-optimal configurations.

The performance loss of zero selection probability configurations is close to 0%. It is also reasonable, because their presence or absence does not affect the choice of top-optimal configurations. So the performance degradation of missing high selection probability configurations is understandable.

At the same time, the performance loss of the selection of the near-optimal configuration is low. Whether it is a configuration with high selection probability or with zero selection probability, the performance loss caused by them is about 5%. Because the near-optimal configurations include the configurations that their execution time is within 110% of the shortest time. The workloads can be recommended to other configurations besides the optimal configuration.

In summary, our ORHRC model can alleviate its item cold start problem.

V. Conclusion and Future Work

In this paper, we design a configuration recommendation model ORHRC to adapt to the cloud-fog orchestrated computing environments. The ORHRC model converts the operating characters of workloads as implicit feedback and the execution time of workloads as the explicit ratings. Then, ORHRC integrates them into a data matrix. Based on the data matrix, ORHRC uses the MF model and NN model to mine the linear and nonlinear relationship between workloads and configurations. Finally, the output of the MF model and NN model are integrated to predict the ranking of configurations for different workloads, so as to give configuration recommendations. The experiment results show that the average hit rate of ORHRC is 24% higher than Micky and 15% higher than Selecta. Besides, our ORHRC model can effectively alleviate cold start problems with SARA.

In the future, we hope to adapt our ORHRC model to the edge computing environment. Containers are more concise than virtual machines and have lower performance overhead. Based on the container technology, edge terminal devices that have fewer resources can be considered to provide configuration options for smaller workloads. At the same time, we plan to make configuration recommendations of heterogeneous clusters for complex and large workloads. Finally, we can greatly expand the applicable workloads range of the model and provide configuration recommendation services for more users.

ACKNOWLEDGMENT

The work of this paper is supported by National Key Research and Development Program of China (2019YFB1405000), National Natural Science Foundation of China under Grant (No. 61873309, No. 61572137, and No. 61728202), and Shanghai Science and Technology Innovation Action Plan Project under Grant (No.19510710500, No.18510760200, and No. 18510732000). This work is partially supported by the project: PCL Future Greater Bay Area Network Facilities for Large scale Experiments and Applications LZC 0019.

REFERENCES

[1] Z. Lv, X. Wang, J. Wu, and P. C. K. Hung, "Instechah: Cost-effectively autoscaling smart computing hadoop cluster in private cloud," *J. Syst. Archit.*, vol. 80, pp. 1–16, 2017.

[2] Z. Wu, Z. Lu, P. C. K. Hung, S. Huang, Y. Tong, and Z. Wang, "Qamec: A qos-driven iovs application optimizing deployment scheme in multimedia edge clouds," *Future Gener. Comput. Syst.*, vol. 92, pp. 17–28, 2019.

[3] C. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 97:1–97:37, 2019.

[4] Z. Lv, N. Wang, J. Wu, and M. Qiu, "Iotdem: An iot big data-oriented mapreduce performance prediction extended model in multiple edge clouds," *J. Parallel Distributed Comput.*, vol. 118, no. Part, pp. 316–327, 2018.

[5] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 90:1–90:33, 2019.

[6] AWS, "Amazon ec2 web instance types for users," https://www.amazonaws.cn/en/ec2/instance-types/.

[7] X. Chen, S. Tang, Z. Lu, J. Wu, Y. Duan, S. Huang, and Q. Tang, "idisc: A new approach to iot-data-intensive service components deployment in edge-cloud-hybrid system," *IEEE Access*, vol. 7, pp. 59 172–59 184, 2019.

[8] A. Xiao, Z. Lu, J. Li, J. Wu, and P. C. K. Hung, "SARA: stably and quickly find optimal cloud configurations for heterogeneous big data workloads," *Appl. Soft Comput.*, vol. 85, 2019.

[9] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *13th USENIX Symposium on Networked Systems Design and Implementation NSDI*, 2016, pp. 363–378.

[10] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 469–482.

[11] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented bayesian optimization for finding the best cloud vm," in *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 660–670.

[12] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach," in *Proceedings of the Symposium on Cloud Computing, SoCC*, 2017, pp. 452–465.

[13] C. Hsu, V. Nair, T. Menzies, and V. W. Freeh, "Micky: A cheaper alternative for selecting cloud instances," in *11th IEEE International Conference on Cloud Computing, CLOUD*, 2018, pp. 409–416.

[14] J. Li, Z. Lu, W. Zhang, J. Wu, H. Qiang, B. Li, and P. C. K. Hung, "SERAC3: smart and economical resource allocation for big data clusters in community clouds," *Future Generation Comp. Syst.*, vol. 85, pp. 210–221, 2018.

[15] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *USENIX Annual Technical Conference, USENIX ATC*, H. S. Gunawi and B. Reed, Eds., 2018, pp. 759–773.

[16] Y. Xu, J. Li, Z. Lu, J. Wu, P. C. Hung, and A. Alelaiwi, "Arvmec: Adaptive recommendation of virtual machines for iot in edge-cloud environment," *Journal of Parallel and Distributed Computing*, vol. 141, pp. 23 – 34, 2020.

[17] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web, WWW*, 2017, pp. 173–182.

[18] Scout, "Large-scale performance data of hadoop and spark on aws," https://github.com/oxhead/scout.

[19] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Workshops Proceedings of the 26th International Conference on Data Engineering, ICDE*, 2010, pp. 41–51.

[20] SparkPerf, "spark-perf: Performance tests for apache spark," https://github.com/databricks/spark-perf/.