Received December 9, 2019, accepted January 7, 2020, date of publication January 20, 2020, date of current version January 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2967790

WebIDE Cloud Server Resource Allocation With Task Pre-Scheduling in IoT Application Development

HUAIJUN WANG[1,2], **JUNHUAI LI**[2], **JUBO TIAN**[2], **AND KAN WANG**[2]
[1]State Key Laboratory of Eco-hydraulics in Northwest Arid Region, Xi'an University of Technology, Xi'an 710048, China
[2]Faculty of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China

Corresponding author: Kan Wang (wangkan@xaut.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1703003, in part by the Research Fund of the State Key Laboratory of Eco-hydraulics in Northwest Arid Region, Xi'an University of Technology, under Grant 2019KJCXTD-6, in part by the National Natural Science Foundation of China under Grant 61971347, Grant 61602375, and Grant 61801379, in part by the Scientific Research Program of Shaanxi Province under Grant 2016KTZDNY01-06 and Grant 2018HJCG-05, in part by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant 2018JQ6057, in part by the Scientific Research Program Funded by Shaanxi Provincial Education Department under Grant 18JK0563, and in part by the Project of Xi'an Science and Technology Planning Foundation under Grant 201805037YD15CG21(4).

ABSTRACT WebIDE is leveraged for IoT application development, which could adapt to the rapid growth of IoT applications and meanwhile facilitate the rapid development. Resource allocation is of vital significance in the WebIDE cloud service system. Existing resource allocation approaches may encounter issues such as unbalanced resource assignments, which could lead to the reduced system resource utilization or extended system response time. Existing methods are typically on the basis of predetermined resource demands for each task, and not applicable to the case that the resource demands are dynamic and unknown. This article predicts the tasks to be performed by the WebIDE cloud service through task pre-scheduling, and then applies the existing resource allocation methods. Firstly, all tasks are classified, based on the execution state, execution operations and WebIDE cloud server resource requirements. Secondly, the grouped tasks are mapped to different system states, with the Markov state transition probability matrix leveraged to model the transition probability between tasks, followed by the prediction model constructed. Finally, integrating task pre-scheduling with ant colony algorithm, WebIDE cloud server resource allocation is carried out. Experiment results show that adding the task prediction model could significantly not only reduce the task response time, but also improve the cloud server resource utilization.

INDEX TERMS Cloud server resource allocation, task pre-scheduling, Markov state transition probability matrix, IoT application development.

I. INTRODUCTION

With With the emergence of visual interaction, cloud server as well as IoT applications, locally executed applications have been migrated to the cloud, and a so-called cloud-based Integrated Development Environment (IDE) (e.g., WebIDE) is gradually becoming a hot spot in the academia [1]. Likewise, in the industry, more and more corporations begin to use WebIDE for application development, and exemplary WebIDE systems include Coding WebIDE, Cloud9, Eclipse che and Codenvy, etc. [2] The WebIDE system relies on cloud servers and browsers to provide users with visual file management and code editing, and supports the one-stop deployment of programming environment and code running. Cloud server, which relies on server cluster, is an application offering functions such as fast computing and secure storage. It is on the basis of distributed computing, parallel computing, network storage and virtualization technologies. Unlike the legacy computing and storage approaches in the PC era, cloud server takes the network as the intermediate medium and turns to executing data storage and computing in the remote cluster service centrer. For users, the cluster service centre is similar to the invisible "cloud" end. In particular, cloud servers are typically with four characteristics, i.e., virtualization, super-scale, high reliability, on-demand service [3]–[5].

The associate editor coordinating the review of this manuscript and approving it for publication was Honggang Wang.

Resource allocation is the key issue in cloud services. Suppose that there exist four physical servers (hosts) with the identical hardware environment and four tasks to be assigned. Then, two illustrative resource allocation schemes could be provided, as shown in Figure 1. Figure 1 (a) prioritizes the host with the most remaining resources for allocation. In this case, task allocation is even, which can guarantee the quality of users, yet with a lower resource utilization. Figure 1 (b) is a physical machine that prioritizes the allocation with the least remaining resources. As such, the resource utilization could be improved, yet resulting in a higher Service-Level Agreement (SLA) violation rate. In particular, if tasks are excessively concentrated on the physical machine (as shown in Figure 1 (c)), then CPU and memory requirements for the host would be high, and there might be a security risk of outage. Therefore, in view of high-concurrency tasks, an efficient resource allocation approach operating on the cloud server is prerequisite to improve resource utilization and reduce task response time.

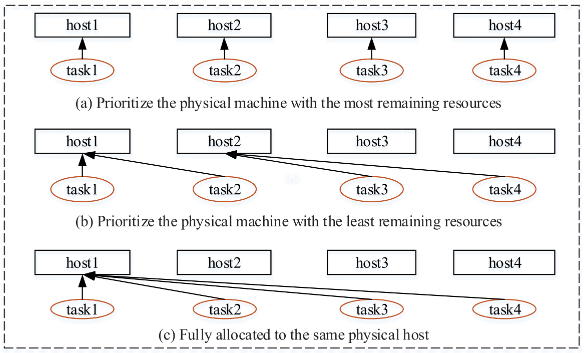

host1
host2
host3
host4
task1
task2
task3
task4
(a) Prioritize the physical machine with the most remaining resources
(b) Prioritize the physical machine with the least remaining resources
(c) Fully allocated to the same physical host

FIGURE 1. Physical machine resource allocation scheme.

Numerous existing works have focused on the resource allocation problem of cloud servers in the WebIDE [6]–[17], by utilizing particle swarm, ant colony or packing optimization algorithms. In reality, the basic idea behind task scheduling is to allocate resources to tasks with the purpose of time loss minimization and performance maximization [6]–[8]. To this end, one scheduling framework was designed to consider applications' I/O performance in terms of acceptable latency, then translating them to associated priority values for disk access [9]. More specially, Agrawal *et al.* [10] compared the resource allocation problem with the boxing one, transformed the server integration modelling into a vector packaging problem, and then proposed a grouping genetic algorithm based on VPC (a vector packing problem with conflicts). Furthermore, Mishra and Sahoo [11] proposed a vector distribution method built on mathematical statistics theory, and Mills *et al.* [12] presented an online boxing distribution approach to solve the resource allocation conflict. Moreover, for the resource scheduling and without violating the SLA protocol, a multi-objective task scheduling algorithm was applied to improve the data centre throughput and reduce the data center cost [13]. In addition, Juarez *et al.* in [14] applied the dynamic energy perception to parallel task scheduling and Bhupesh *et al.* in [18] propose a self-optimized energy efficient resource management strategy, while Alsarhan *et al.* in [15] studied the adaptive allocation and configuration of resources in a multi-service cloud environment. Besides, to balance the workload among all virtual machines with resilient resource provisioning and de-provisioning, one dynamic k-interval based scheduling algorithm was proposed in [16]. Moreover, the comparative experimental performance study in [19] demonstrate the space shared technique for cloud resource scheduling is much beneficial as compared to the time shared technique. What is more, Priya *et al.* in [20] constructs a Fuzzy-based multidimensional resource scheduling model to avoid underutilization and overutilization of resources, improving latency time for each class of request.

All aforementioned works [6], [7], [9]–[16] solved the cloud server resource allocation under the premise of task predetermination, i.e., the arrival time, the execution time and required resources for each task are deterministic and predetermined. Nevertheless, in the practical operation of cloud servers, there exist numerous uncertainties with respect to tasks [21]–[23], e.g., dynamic arrival and stochastic execution time, which would result in a lowered prediction accuracy. Therefore, it is prerequisite to study the resource allocation of cloud servers under the premise of task uncertainties. If the arrival time, execution time and required resources of the task can be known, the resource allocation efficiency will be improved. Then we propose a method to improve the existing resource allocation algorithm by predicting the undetermined tasks. The main contributions of this paper can be summarized as follows:

- We propose a task prediction model to predict the succeeding tasks to be executed in the WebIDE system, with the Markov state transition matrix involved. The Markov state transfer matrix can describe the state transition between tasks in the webIDE system.
- To exploit the system resources more efficiently, a resource allocation approach integrating both colony algorithm and task prediction model is presented. After the task prediction, the predicted one is transferred into the virtual queue for task pre-scheduling, which contributes to the system resource preparation. Then, the ant colony algorithm could be utilized to allocate resources in the cloud server optimally with the least time to complete all tasks.
- Extensive simulations are conducted with diverse system parameters to show the effectiveness of proposed approach. It is revealed that the task prediction model could significantly not only reduce the task response time, but also improve the cloud server resource utilization.

The remainder of this paper is organized as follows. In Section II, we present the cloud server resource allocation model based on task pre-scheduling. In Section III, we formulate the task classification in WebIDE system. In Section IV,

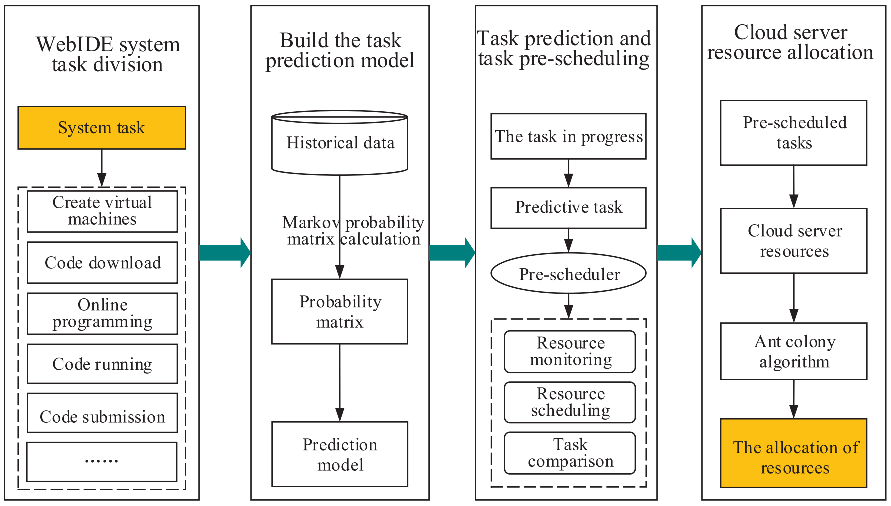


FIGURE 2. Cloud server resource allocation model based on task pre-scheduling.

we construct the task prediction model based on Markov state transition probability matrix. In Section V, we solve the task pre-scheduling based on task prediction model. In Section VI, we propose the cloud server resource allocation with task prediction involved. In section VII, experiments results are shown to reveal the effectiveness of proposed approach. Section VIII concludes this paper with future work.

II. CLOUD SERVER RESOURCE ALLOCATION MODEL BASED ON TASK PRE-SCHEDULING

In the proposed WebIDE system, the cloud server resource allocation model based on task pre-scheduling consists of four modules shown in Figure 2, i.e., task division, task prediction, task pre-scheduling and resource allocation, as follows:

- WebIDE system task division: the tasks in the WebIDE system are categorized into groups on the basis of execution state, execution operations, as well as cloud server resource requirements.
- Task prediction model construction: the grouped tasks are mapped to different system states. In particular, Markov state transition matrix [24] is leveraged to model the transition probability between tasks, followed by the prediction model constructed.
- Task prediciton and pre-scheduling: both the system task queue and virtual machine local task queue are involved in the pre-scheduler, and the predicted succeeding task is placed into the virtual machine local task queue for pre-scheduling. The cloud server resource information is collected through the resource monitoring module in the scheduler. Meanwhile, the resource scheduling module acquires resource consumption data required by historical tasks analogously, and combines the resource monitoring module to collect resource information for scheduling, so as to make resource available in advance for prediction.
- Cloud server resource allocation: pre-scheduled tasks, cloud server resources and other information are introduced into ant colony algorithm, which has been widely utilized in the cloud server resource allocation, and could significantly reduce both the task response time and resource consumption when emergencies are triggered.

III. WEBIDE SYSTEM TASK TRANSITION MODEL

A. DEFINITIONS

The task model of WebIDE system is defined as

$$T = (DC, ATs, DIs), \tag{1}$$

where DC is the function description, ATs is the action set, i.e., $ATs = \{AT_1, AT_2, \ldots, AT_m\}$, and DIs is the set of resource items, i.e., $DIs = \{DI_1, DI_2, \ldots, DI_n\}$. Next, the task granularity model is defined as

$$P = \{AT, DI\}, \tag{2}$$

where AT and DI are the actions taken by the task and resource item, respectively. When action and resource items are identical in the task granularity, they are supposed to be with the same task granularity.

In line with the general function of WebIDE system, the ATs and DIs of action set in task model are instantiated as

$$\begin{aligned} ATs = \{&\text{Virtual machines and engineering creation,}\\ &\text{Code download, Online programming,}\\ &\text{Code execution, Code submition}\}, \end{aligned} \tag{3}$$

and

$$\begin{aligned} DIs = \{&\text{CPU, Memory, Compile environment,}\\ &\text{Project files, Code, Network}\}, \end{aligned} \tag{4}$$

respectively.

In addition, T_a and T_b are defined as the sets of tasks before and after partition, respectively. Then, T_a is categorized into T_b according to the task granularity as follows:

$$T_a = \begin{cases} T_{b1} = \{\text{Virtual machines and engineering creation,} \\ \qquad (\text{CPU, Memory, Network})\} \\ T_{b2} = \{\text{Code download, (Network, Hardware)}\} \\ T_{b3} = \{\text{Online programming, (Network)}\} \\ T_{b4} = \{\text{Code execution, (CPU, Memory, Hardware)}\} \\ T_{b5} = \{\text{Code submition, (Hardware, Network)}\}. \end{cases} \tag{5}$$

B. DETERMINATION OF TASK DEPENDENCIES

In line with ATs instantiated in Equation (3) and the requirements of CPU and memory in the WebIDE system, the TY dependent set after task action can be obtained as

$$TY = \begin{cases} TY_2 = \{TY_1, TY_5\} \\ TY_3 = \{TY_1, TY_2, TY_5\} \\ TY_4 = \{TY_1, TY_3\} \\ TY_5 = \{TY_2, TY_3, TY_4\}. \end{cases} \tag{6}$$

According to Equations (5) and (6), the dependent set R between partitioned tasks is as

$$R = \begin{cases} T_{b2} = \{T_{b1}, T_{b5}\} \\ T_{b3} = \{T_{b1}, T_{b2}, T_{b5}\} \\ T_{b4} = \{T_{b1}, T_{b3}\} \\ T_{b5} = \{T_{b2}, T_{b3}, T_{b4}\}. \end{cases} \tag{7}$$

C. TASK TRANSITION MODEL CONSTRUCTION

By analysing the dependency set R item by item, the task transition model can be acquired following the dependencies between tasks as in Figure 3. As revealed from Figure 3, tasks can be translated to each other in probability. The transition probability between tasks is described by the two-dimensional matrix, with each element $TM[i][j]$ representing the probability of task i transitioned into task j. In the transition probability matrix, the value of 0 indicates that there is no transition between two tasks. Meanwhile, the sum of each row or column is equal to 1. It should be noted that, the performed task corresponds to the system execution state as

$$T_{bi} \Leftrightarrow S_i, \tag{8}$$

which could represent the mapping of tasks and system execution states (8). Therefore, the study on tasks can be transformed into that on the execution state. It follows that the state transition probability matrix can be solved for the system based on Markov state transition probability matrix.

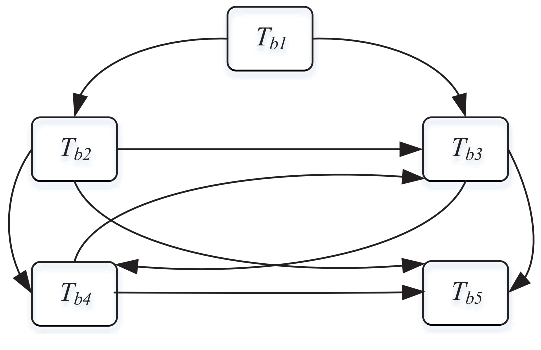


FIGURE 3. Task transition model.

IV. TASK PREDICTION MODEL CONSTRUCTION

In this section, the Markov state transition probability matrix is analysed and each task is regarded as one state in the system operation; thus, the probability matrix could indicate the transition between each task in the system operation.

A. MARKOV TRANSITION PROBABILITY

Suppose there are n incompatible system states, and the state space is defined as $\mathbf{I} = \{1, 2, ..., n\}$. Then Markov one-step state transition probability can be defined as

$$P_{ij}(m) = P\{S(m+1) = j | S(m) = i\}, \quad i, j \in \mathbf{I}, \tag{9}$$

where P_{ij} is the probability that the system changes from state i to state j in one step. Then, Markov k-step state transition probability matrix is defined as

$$P^k_{ij}(m) = P\{S(m+k) = j | S(m) = i\}, \quad i, j \in \mathbf{I}, k \geq 1, \tag{10}$$

where P^k_{ij} is the probability that the system changes from state i to j in k steps. If the system state can be transitioned in k steps, then it can be described by the k-step transition matrix, denoted as $\mathbf{P}^{(k)}$, as

$$\mathbf{P}^{(k)} = \mathbf{P}^{(k-1)} \cdot \mathbf{P} = \mathbf{P}^k, \tag{11}$$

which indicates that, the k-step transition matrix changes again on the basis of previous transition and thus is the k-power of one-step transition matrix.

B. STATE TRANSITION PROBABILITY MATRIX DERIVATION

Assuming that there exist a total of n execution states in the system, the initial system state vector turns out to be

$$\mathbf{s}(0) = [s_{01}, \ldots, s_{0j}, \ldots, s_{0n}], \tag{12}$$

where s_{0j} represents the initial probability that the system lies in state j. Then, the probability that the system lies in state j after k-step transition is denoted as s_{kj}, and the state vector after k-step transition becomes

$$\mathbf{s}(k) = [s_{k1}, \ldots, s_{kj}, \ldots, s_{kn}]. \tag{13}$$

Then, the state i after k-step transition is denoted as s_{ki}, i.e.,

$$\mathbf{S} = \begin{bmatrix} s_{11} & \cdots & s_{1j} & \cdots & s_{1n} \\ s_{21} & \cdots & s_{2j} & \cdots & s_{2n} \\ . & \cdots & \cdots & \cdots & . \\ . & \cdots & s_{ij} & \cdots & . \\ . & \cdots & \cdots & \cdots & . \\ s_{(k-1)1} & \cdots & s_{(k-1)j} & \cdots & s_{(k-1)n} \\ s_{k1} & \cdots & s_{kj} & \cdots & s_{kn} \end{bmatrix}. \tag{14}$$

As shown in Equation (14), state s_{ki} could be obtained by a k-step transition from any state, and the first row of the matrix represents the state at which the transition occurs.

Then, suppose there exists a state transition probability matrix $\mathbf{P}$ as

$$\mathbf{P} = \begin{bmatrix} P_{11} & \dots & P_{1j} & \dots & P_{1n} \\ P_{21} & \dots & P_{2j} & \dots & P_{2n} \\ . & \dots & \dots & \dots & . \\ . & \dots & P_{ij} & \dots & . \\ . & \dots & \dots & \dots & . \\ P_{(n-1)1} & \dots & P_{(n-1)1j} & \dots & P_{(n-1)n} \\ P_{n1} & \dots & P_{nj} & \dots & P_{nn} \end{bmatrix}. \quad (15)$$

where P_{ij} represents the probability of state i transitioning to state j. As such, state s_{21} could obtained by $\mathbf{s}(1)$ (all the states after one-step) multiplied by $[P_{11}, P_{21}, \dots, P_{n1}]^T$, i.e., $s_{21} = \mathbf{s}(1) \times [P_{11}, P_{21}, \dots, P_{n1}]^T$. Likewise, s_{2n} is acquired by state $\mathbf{s}(1)$ multiplied by $[P_{1n}, P_{2n}, \dots, P_{nn}]^T$, i.e., $s_{2n} = \mathbf{s}(1) \times [P_{1n}, P_{2n}, \dots, P_{nn}]^T$. Therefore, we have

$$\begin{cases} s_{11} \times P_{11} + s_{12} \times P_{21} + \dots + s_{1n} \times P_{n1} = s_{21} \\ s_{11} \times P_{12} + s_{12} \times P_{22} + \dots + s_{1n} \times P_{n2} = s_{22} \\ \dots\dots \\ s_{11} \times P_{1n} + s_{12} \times P_{2n} + \dots + s_{1n} \times P_{nn} = s_{2n} \\ \dots\dots \\ s_{(k-1)1} \times P_{11} + s_{(k-1)2} \times P_{21} + \dots + s_{(k-1)n} \times P_{n1} = s_{k1} \\ s_{(k-1)1} \times P_{12} + s_{(k-1)2} \times P_{22} + \dots + s_{(k-1)n} \times P_{n2} = s_{k2} \\ \dots\dots \\ s_{(k-1)1} \times P_{1n} + s_{(k-1)2} \times P_{2n} + \dots + s_{(k-1)n} \times P_{nn} = s_{kn}. \end{cases} \quad (16)$$

which can be written more compactly as

$$\begin{bmatrix} \mathbf{P}^T & & & \\ & \mathbf{P}^T & & \\ & & \dots & \\ & & & \mathbf{P}^T \end{bmatrix} \times \begin{bmatrix} \mathbf{s}^T(1) \\ \mathbf{s}^T(2) \\ \dots \\ \mathbf{s}^T(n-1) \end{bmatrix} = \begin{bmatrix} \mathbf{s}^T(2) \\ \mathbf{s}^T(3) \\ \dots \\ \mathbf{s}^T(n) \end{bmatrix}. \quad (17)$$

Also, the sum of probability moving from one state to all other states is equal to one, namely,

$$\begin{cases} P_{11} + P_{12} + \dots + P_{1n} = 1 \\ P_{21} + P_{22} + \dots + P_{2n} = 1 \\ \dots\dots \\ P_{n1} + P_{n2} + \dots + P_{nn} = 1. \end{cases} \quad (18)$$

Integrating Equation (18) with the historical data, the state transition probability matrix $\mathbf{P}$ can be finally acquired.

C. TASK PREDICTION MODEL CONSTRUCTION

The Markov task prediction model is constructed on the state transition probability matrix, which eventually turns out to be:

$$\begin{aligned} &\mathbf{s}(k) \\ &= \mathbf{s}(k-1) \times \mathbf{P} \\ &= \mathbf{s}(k-1) \times \begin{bmatrix} P_{11} & \dots & P_{1j} & \dots & P_{1n} \\ P_{21} & \dots & P_{2j} & \dots & P_{2n} \\ . & \dots & \dots & \dots & . \\ . & \dots & P_{ij} & \dots & . \\ . & \dots & \dots & \dots & . \\ P_{(n-1)1} & \dots & P_{(n-1)j} & \dots & P_{(n-1)n} \\ P_{n1} & \dots & P_{nj} & \dots & P_{nn} \end{bmatrix} \\ &= \mathbf{s}(0) \times \mathbf{P}^k \\ &= \mathbf{s}(0) \times \begin{bmatrix} P^k_{11} & \dots & P^k_{1j} & \dots & P^k_{1n} \\ P^k_{21} & \dots & P^k_{2j} & \dots & P^k_{2n} \\ . & \dots & \dots & \dots & . \\ . & \dots & P^k_{ij} & \dots & . \\ . & \dots & \dots & \dots & . \\ P^k_{(n-1)1} & \dots & P^k_{(n-1)j} & \dots & P^k_{(n-1)n} \\ P^k_{n1} & \dots & P^k_{nj} & \dots & P^k_{nn} \end{bmatrix}. \end{aligned} \quad (19)$$

It should be noted that, if the initial state $\mathbf{s}(0)$ is known, then $\mathbf{s}(0) \times \mathbf{P}^k$ can be used to acquire the system execution state given parameter k. By taking into account Equations (7), (8) and (19), the system execution state prediction model can be translated into the task prediction model with the initial task vector defined as

$$\mathbf{t}(0) = [\mathbf{t}_{b1}(0), \dots, \mathbf{t}_{bj}(0), \dots, \mathbf{t}_{bn}(0)]. \quad (20)$$

Finally, the task transition prediction model derived on the system execution state prediction one can be represented as

$$\mathbf{t}(k) = \mathbf{t}(k-1) \times \mathbf{P} = \mathbf{t}(0) \times \mathbf{P}^k. \quad (21)$$

V. PREDICTION-BASED TASK PRE-SCHEDULING

For simplicity, the task being performed on the virtual machine is called "VM execution task", and the predicted task that would be scheduled on the virtual machine is called "VM prediction task". Nevertheless, during the execution of "VM execution task", the triggered tasks are termed as "VM task 1", "VM task 2", ..., "VM task n", which are readily stored in the system task queue, waiting to be called for execution. Among them, there is only one system task queue in the whole system, and the tasks actually triggered by users in the system are stored in the task queue. Different virtual machines have their proprietary virtual machine task queues, and both the tasks that are being executed and that are to be executed need to be stored, as shown in Figure 4.

In the task pre-scheduling model in Figure 4, the system task queue is built in both the entire service and control module, comprising all tasks that need to be executed by the virtual machine (including both new arrival tasks and waiting

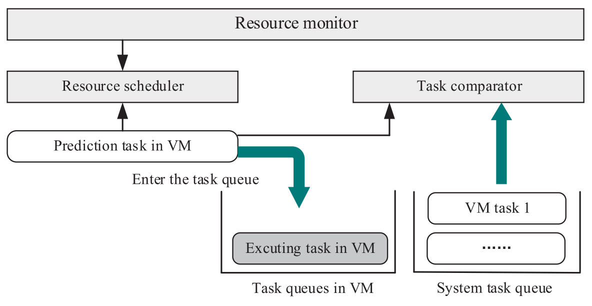


FIGURE 4. Physical machine resource allocation scheme.

tasks). VM's task queue is virtual machine VM task queue, includes both tasks that are being executed and those to be performed; the resource monitor supervises resource usage across the cloud server, while the resource scheduler conducts resource scheduling to prepare resources for the execution of predicted tasks. Moreover, the task comparator is responsible for comparing the predicted task with that belonging to the virtual machine (to be executed in the system task queue). The detailed task pre-scheduling process is as follows:

- On the basis of prediction model in Section IV-C, the next task is predicted on the task queue in VM. Then the predicted task is called into the task queue as the one to be executed.
- The resource information required for the execution of historical tasks (which are analogous to the "VM prediction task") is acquired for the execution of "VM prediction task". In particular, the resource scheduler adjusts resources in line with both the resource information required by the "VM prediction task" and the resource monitoring information from the resource monitor, so as to prepare resources for the execution of predicted one.
- The task comparator is responsible for comparing the "VM prediction task" and "VM task 1" in the system task queue. The identical result indicates an accurate prediction, while the opposite one necessitates that the "predicted task in VM" is discarded and "VM task 1" in the system task queue is called into the virtual machine VM task queue.

In particular, if the task prediction accuracy is sufficiently high, then the next task can be called into the virtual machine task queue beforehand. Accordingly, cloud server resource scheduling also works in advance to prepare resources for the next task execution. Thus, the overall response time of task execution could be reduced and meanwhile the resource utilization is improved.

VI. CLOUD SERVER RESOURCE ALLOCATION

An efficient resource allocation in cloud servers is related to the scheduling approach. If the cloud server can be aware of pending tasks beforehand, then it can reduce task response time and resource consumption. Ant colony algorithm, which could collect discrete resource information and reach the optimal solution of resource allocation, has been widely utilized in cloud servers. In particular, the ant colony algorithm could speed up the evolution through positive feedbacks, which is consistent with the real-time demand of cloud servers. The ant colony algorithm is depicted as follows:

- In line with the pheromone concentration and heuristic information on the route, the next route is chosen;
- Upon arrival at the destination, the associated pheromone information is released and the pheromone on the route is updated.

The pheromone concentration is described by the ant route selection transition function. Integrated with the heuristic information, the next route of the k-th ant is selected as

$$P_{i,j}^{k} = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha} \times [\eta(i,j)]^{\beta}}{\sum_{s \in r_k} [\tau(i,s)]^{\alpha} \times [\eta(i,s)]^{\beta}}, & j \in r_k \\ 0, & j \notin r_k \end{cases} \tag{22}$$

where r_k represents the k-th path of all routes through which an ant can pass at point i. $\eta(i,j)$ represents the heuristic information, in general expressed as the reciprocal of distance between point i and j. In particular, the closer the distance is, the larger the heuristic information value would be, and the more decisive it would be to screen the next point. $\tau(i,j)$ is the pheromone concentration on the path from point i to j. α and β are respectively utilized to adjust the weights of pheromone concentration and heuristic information in ant route selection function. When α is 0, the route transition function becomes a simple random greedy function; and when β is 0, the function completely depends on the pheromone concentration to determine the next route, which would lead to the occurrence of extreme cases in practical problems. As such, it might result in a large error between the route constructed and the actual one.

Then, the pheromone update function is expressed as

$$\begin{cases} \tau_{i,j}(t+1) = \rho\tau_{i,j}(t) + \Delta\tau_{i,j}, & 0 < \rho < 1 \\ \Delta\tau_{i,j} = \sum_{k=1}^{n} \Delta\tau_{i,j}^{k}, \end{cases} \tag{23}$$

where n represents the ant number, and ρ $(0 < \rho < 1)$ is the evaporation concentration of pheromones. Moreover, $\Delta\tau_{i,j}$ is the amount of pheromones released by the k-th ant along the route between point i and j.

As in (23), after updating, the amount of pheromones would reduce due to evaporation. Only the amount of pheromones in the path that has been visited by ants would increase or remains the same. Moreover, changes are related to the length of routes that the ant has traveled. The shorter the distance is, the more pheromones would be released, which would in turn increase the amount of pheromones on the route. Conversely, for a sufficiently long route, the amount of pheromones decreases or remains basically unchanged.

Next, integrating ant ant colony algorithm with task pre-scheduling, the cloud server resource allocation can be described below. It is assumed that resource $C = \{C_1, C_2, \ldots C_N\}$ is composed of different computing capacities, and $T = \{T_1, T_2, \ldots T_K\}$ represents the tasks to be processed on each resource node. First, the executing task is captured and the next one is predicted with the prediction model in Section IV-C. Secondly, the predicted task is called into the virtual machine task queue for task pre-scheduling. Thirdly, ant colony algorithm is leveraged to allocate resources for predicted tasks. The cloud server resource allocation could be summarized in detail as follows:

1) Set the coefficients in the transition matrix of ant colony algorithm according to [**?**];
2) Assign tasks to the cloud server node randomly;
3) Following (23), update the pheromone concentration for the task associated with each cloud server node;
4) As in (22), choose the cloud server node associated with the task;
5) Calculate the probability that the task is consistent with the cloud server node, and add results to the next loop queue for comparison;
6) The number of cycles is increased by 1. If k is less than the total number of tasks, then the pheromone is updated as in (23) and the loop starts from step 3;
7) Find the optimal solution and output the selection results, which are the tasks to be executed.

VII. EXPERIMENT AND ANALYSIS

Some existing works have conducted experiments on resource allocation of cloud servers. For instance, an allocation method with both ant colony and genetic algorithms was proposed with 8 virtual machines in the Matlab simulation. The task number increases from 20 to 100, and the execution time of each task is random. Through dynamic load balancing algorithm, services are deployed on the cloud server to detect the utilization rate of virtual machine CPU, memory and other resources. A task scheduling algorithm with QoS classification was proposed in [25], with CloudSim simulating multiple virtual machines and tasks. In particular, CloudSim is utilized to simulate 200 virtual machines. In addition, genetic algorithm, ant colony algorithm and an improved one are respectively applied, with the number task increased from 100 to 500.

A. EXPERIMENT 1: PREDICTION MODEL ACCURACY AND RESPONSE TIME

Firstly, the task history information is collected. Then, the probability matrix is calculated with Markov transition probability, and the next task is predicted by constructing the prediction model. The task and cycle number are set to be 5 and 20, respectively. Figure 5 reveals the task prediction accuracy P_t in the task prediction stage, which can be obtained as

$$P_t = N_p/N_t, \tag{24}$$

where N_p represents the number of tasks predicted correctly, and N_t denotes the number of all prediction tasks. As can be seen from Figure 5, the accuracy decreases with the growth of task prediction times. Moreover, the adjustment frequency is set to be 5, that is to say, the adjustment is prerequisite when the prediction number reaches 5. Furthermore, the state

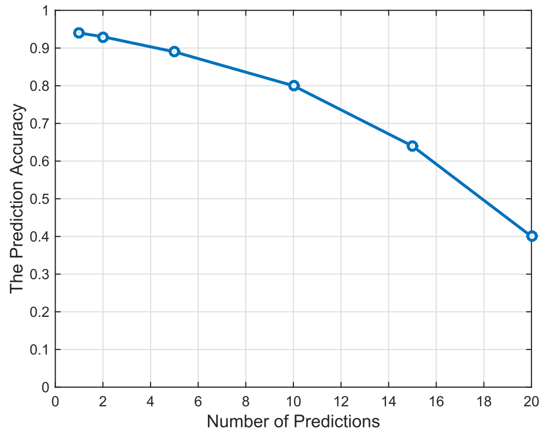


FIGURE 5. Accuracy rate of task prediction.

transition probability matrix is recalculated according to Equations (15) and (19). In addition, the prediction model is adjusted for task prediction with the prediction accuracy shown in Figure 6.

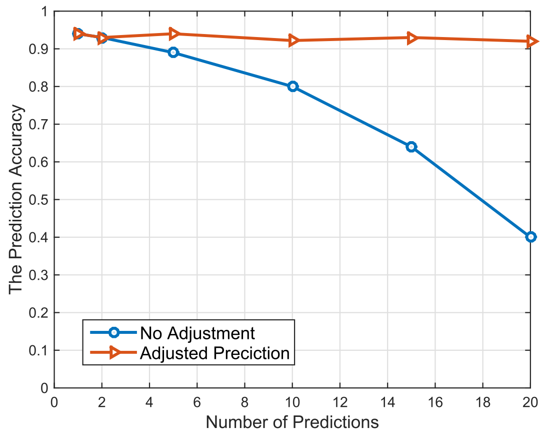


FIGURE 6. Adjusted and non-adjusted accuracy rate.

As can be seen from Figure 6, when the prediction model is adjusted after 5 consecutive task predictions, the prediction accuracy could be significantly improved. Nevertheless, frequent adjustments would in turn increase the task response time. In particular, the adjustment frequency is set as 1, 2, 5, 10 and 20, respectively, with their associated results shown in Figure 7. Meanwhile, the average task response time within each adjustment and the average task response time after 20 task predictions are shown in Table 1. From Figure 7 and Table 1, it follows that the minimum response time lies in the adjustment frequency of 5.

B. EXPERIMENT 2: TASK RESPONSE TIME WITH DIFFERENT TASK NUMBERS

Experiments are conducted to show the difference in task response time of different approaches. As the task number gradually increases from 5 to 200, the average task response time for two different algorithms is shown in Figure 8. As can

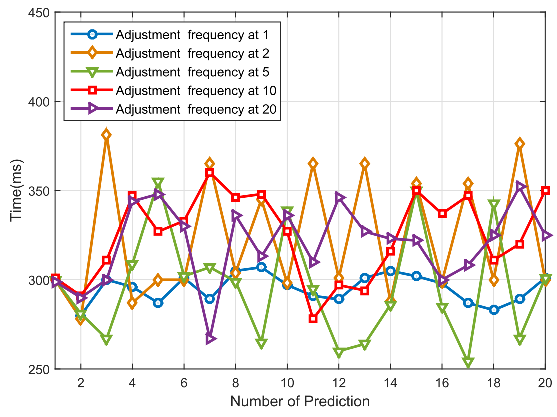


FIGURE 7. Task response time with different frequencies.

TABLE 1. The average task response time with different frequency.

Frequency (time)	Average task response time in one adjustment cycle	Average time (ms)
1	300, 280, 300, 296, 287, 301, 289, 305, 307, 297, 291, 289, 301, 305, 302, 298, 287, 283, 289, 301	295.4
2	287, 333.5, 332.5, 332.3, 323, 329, 327, 326.5, 328, 339	325.78
5	300.6, 298.9, 290.2, 289.8	294.88
10	328.6, 318.8	323.7
20	318.55	318.55

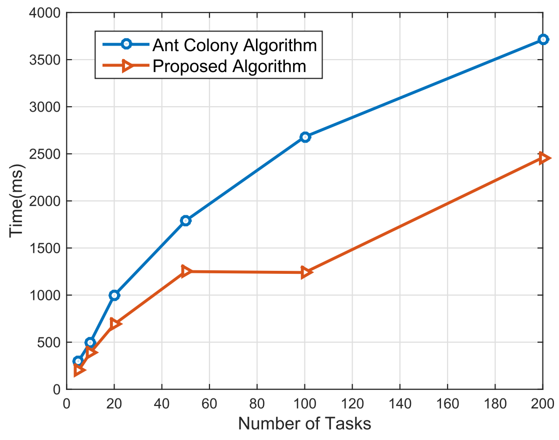


FIGURE 8. The average response time of tasks with pre-scheduling and non pre-scheduling.

be seen from Figure 8, with the increase of task number, the average task response time with both task prediction and pre-scheduling become significantly lower than that of ant colony algorithm.

C. EXPERIMENT 3: RESOURCE UTILIZATION BASED ON CLOUDSIM

Cloud experiment environment is simulated using CloudSim. With the task number increased from 5 to 500, the resource allocation is carried out by our proposed approach as well as ant colony algorithm, respectively. The CPU and memory utilization results are shown in Figures 9 and 10, respectively.

In this part, 500 tasks are randomly generated. As shown in Figures 9 and 10, as the task number increases from 50 to 100, both CPU and memory utilization increase significantly and then converge. In particular, with the increase of task number from 50 to 100, some tasks with high CPU and memory requirements (e.g., code running) are added. Thus, there exist differences in hardware resource demand in the WebIDE system for different tasks. In addition, given the task number, our proposed approach requires less resources than the ant colony algorithm.

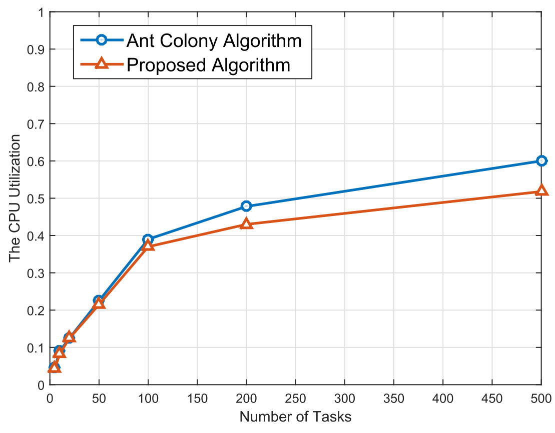


FIGURE 9. CPU utilization.

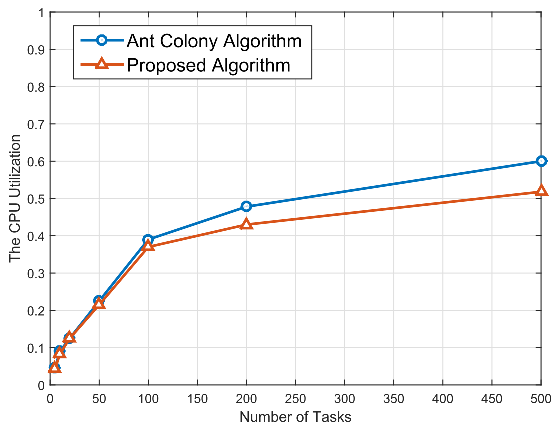


FIGURE 10. Memory utilization.

VIII. CONCLUSION

In this paper, existing cloud server resource allocation approaches were analyzed, the current cloud server resource allocation algorithm is based on the premise of request task determination. The method proposed in this paper predicts the tasks to be executed by the WebIDE cloud service, preschedules the tasks after the prediction, and then applies the existing resource allocation method to allocate the resources of the WebIDE cloud server. Experiments showed that if the resource allocation of the WebIDE cloud server is carried out after task pre-scheduling, then the average task response time can be effectively reduced and the resource utilization is improved.

REFERENCES

[1] N. Sidhanth, S. Sanjeev, S. Swettha, and R. Srividya, "A next generation IDE through multi tenant approach," *Int. J. Inf. Electron. Eng.*, vol. 4, no. 1, pp. 27–30, Jan. 2014.

[2] G. Fylaktopoulos, G. Goumas, M. Skolarikis, A. Sotiropoulos, D. Athanasiadis, and I. Maglogiannis, "CIRANO: An integrated programming environment for multi-tier cloud based applications," *Procedia Comput. Sci.*, vol. 68, pp. 42–52, 2015.

[3] T. Shin, H.-T. Jeon, and J. Byun, "Developing nontrivial standby power management using consumer pattern tracking for on-demand appliance energy saving over cloud networks," *IEEE Trans. Consum. Electron.*, vol. 62, no. 3, pp. 251–257, Aug. 2016.

[4] C. Li, Y. C. Liu, and X. Yan, "Optimization-based resource allocation for software as a service application in cloud computing," *J. Scheduling*, vol. 20, no. 1, pp. 103–113, Feb. 2017.

[5] N. Zhao, F. Cheng, F. R. Yu, J. Tang, Y. Chen, G. Gui, and H. Sari, "Caching UAV assisted secure transmission in hyper-dense networks based on interference alignment," *IEEE Trans. Commun.*, vol. 66, no. 5, pp. 2281–2294, May 2018.

[6] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Recent advancements in resource allocation techniques for cloud computing environment: A systematic review," *Cluster Comput.*, vol. 20, no. 3, pp. 2489–2533, Sep. 2017.

[7] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019.

[8] N. Zhao, Y. Cao, F. R. Yu, Y. Chen, M. Jin, and V. C. M. Leung, "Artificial noise assisted secure interference networks with wireless power transfer," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1087–1098, Feb. 2018.

[9] N. Jain and J. Lakshmi, "PriDyn: Enabling differentiated I/O services in cloud using dynamic priorities," *IEEE Trans. Services Comput.*, vol. 8, no. 2, pp. 212–224, Mar. 2015.

[10] S. Agrawal, S. K. Bose, and S. Sundarrajan, "Grouping genetic algorithm for solving the serverconsolidation problem with conflicts," in *Proc. 1st ACM/SIGEVO Summit Genetic Evol. Comput. (GEC)*, Jun. 2009, pp. 1–8.

[11] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, Jul. 2011, pp. 275–282.

[12] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand clouds," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2011, pp. 91–98.

[13] A. V. Lakra and D. K. Yadav, "Multi-objective tasks scheduling algorithm for cloud computing throughput optimization," *Procedia Comput. Sci.*, vol. 48, pp. 107–113, Dec. 2015.

[14] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 257–271, Jan. 2018.

[15] A. Alsarhan, A. Itradat, A. Y. Al-Dubai, A. Y. Zomaya, and G. Min, "Adaptive resource allocation and provisioning in multi-service cloud environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 31–42, Jan. 2018.

[16] M. Kumar and S. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Comput. Electr. Eng.*, vol. 69, pp. 395–411, Jul. 2018.

[17] X. Xie, K. Xu, and X. Wang, "Cloud computing resource scheduling based on improved differential evolution ant colony algorithm," in *Proc. Int. Conf. Data Mining Mach. Learn. (ICDMML)*, Apr. 2019, pp. 171–177.

[18] B. K. Dewangan, A. Agarwal, M. Venkatadri, and A. Pasricha, "Self-characteristics based energy-efficient resource scheduling for cloud," *Procedia Comput. Sci.*, vol. 152, pp. 204–211, 2019.

[19] P. Lakkadwala and P. Kanungo, "Performance evaluation of time shared and space shared techniques for cloud resource scheduling," in *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, Mar. 2019, pp. 1–6.

[20] V. Priya, C. Sathiya Kumar, and R. Kannan, "Resource scheduling algorithm with load balancing for cloud service provisioning," *Appl. Soft Comput.*, vol. 76, pp. 416–424, Mar. 2019.

[21] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *J. Syst. Softw.*, vol. 99, pp. 20–35, Jan. 2015.

[22] S. Seth and N. Singh, "Dynamic threshold-based dynamic resource allocation using multiple VM migration for cloud computing systems," in *Proc. Int. Conf. Inf., Commun. Comput. Technol. (ICICCT)*, Oct. 2017, pp. 106–116.

[23] N. Zhao, X. Liu, F. R. Yu, M. Li, and V. C. M. Leung, "Communications, caching, and computing oriented small cell networks with interference alignment," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 29–35, Sep. 2016.

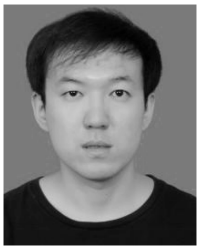
[24] L. Zhang, J. Ai, B. Jiang, H. Lu, and X. Li, "Saliency detection via absorbing Markov chain with learnt transition probability," *IEEE Trans. Image Process.*, vol. 27, no. 2, pp. 987–998, Feb. 2018.

[25] B. L. Pan, Y. P. Wang, H. X. Li, and J. Qian, "Task scheduling and resource allocation of cloud computing based on QoS," *Adv. Mater. Res.*, vols. 915–916, pp. 1382–1385, Apr. 2014.

HUAIJUN WANG received the B.Sc. and M.Sc. degrees in computer science from the Xi'an University of Technology, in 2005 and 2010, respectively, and the Ph.D. degree from Northwest University, Xi'an, China, in 2014. He is currently a Lecturer with the Xi'an University of Technology. His research interests include application and security of CPS and edge calculation based on application drive.

JUNHUAI LI received the B.S. degree in electrical automation from the Shaanxi Institute of Mechanical Engineering of China, Xi'an, in 1992, the M.S. degree in computer application technology from the Xi'an University of Technology of China, Xi'an, China, in 1999, and the Ph.D. degree in computer software and theory from the Northwest University of China, Xi'an, in 2002. He is currently a Professor with the School of Computer Science and Engineering, Xi'an University of Technology. His research interests include the Internet of Things technology and network computing.

JUBO TIAN received the B.Sc. degree in computer science from the Shaanxi University of Science and Technology, in 2014, and the M.Sc. degree from the Xi'an University of Technology, in 2017. Her research interests are the Internet of Things technology and artificial intelligence techniques.



KAN WANG received the B.S. degree in broadcasting and television engineering from the Zhejiang University of Media and Communications, Hangzhou, China, in 2009, and the Ph.D. degree in military communications from the State Key Laboratory of ISN, Xidian University, Xi'an, China, in 2016. From October 2014 to October 2015, he was with Carleton University, Ottawa, ON, Canada, as a Visiting Scholar funded by the China Scholarship Council (CSC). Since March 2017, he has been with the School of Computer Science and Engineering, Xi'an University of Technology, Xi'an. His current research interests include 5G cellular networks, resource management, and massive IoT.