Hindawi
Mobile Information Systems
Volume 2022, Article ID 1849877, 10 pages
https://doi.org/10.1155/2022/1849877

Research Article

Efficient Delay-Aware Task Scheduling for IoT Devices in Mobile Cloud Computing

Chenghou Jin,[1] Jiajie Xu,[1] Yusen Han,[1] Jintao Hu,[1] Ying Chen ,[1] and Jiwei Huang [2]

[1]*School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China*
[2]*Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing, Beijing 102249, China*

Correspondence should be addressed to Ying Chen; chenying@bistu.edu.cn

Received 12 May 2022; Revised 14 June 2022; Accepted 29 June 2022; Published 15 July 2022

Academic Editor: Adarsh Kumar

The development of Internet of Things (IoT) technology depends on technologies such as high-efficiency storage and high computing power. Mobile cloud computing (MCC) technology will be an important foundation for the development of IoT. The efficient scheduling of tasks in IoT devices in MCC environment is challenging. The requirements for task scheduling in MCC are becoming more and more complex. As the core problem in MCC, task scheduling aims to allocate tasks reasonably, achieve optimal scheduling strategies, and complete tasks effectively. In this paper, efficient delay-aware task scheduling algorithm (EDTSA) is proposed, with the optimization goal of minimizing task running time. The matching of tasks and virtual machines is modeled as a bipartite graph. The problem is divided into multiple subproblems to solve the optimal solution separately. The combined solution is used as the initial solution of the local search algorithm. The efficiency of the local search depends on the quality and nature of the initial solution. We can generate multiple initial solutions according to different division criteria. The initial solution is the combination of the optimal solutions of the subproblems, so the quality of the initial solution has been greatly improved and generating multiple initial solutions according to the division can reduce the probability of falling into the local optimal solution. This algorithm also divides the neighborhood to reduce unnecessary searches. Finally, we verify the efficiency and practicability of the algorithm through experiments.

1. Introduction

Mobile cloud computing (MCC) and Internet of Things (IoT) are two hot emerging concepts, and at the same time, they are inseparable. IoT devices generate a large number of tasks at different times and schedule or allocate these tasks, which requires cloud computing with large-scale computing capabilities. Documents [1–4] all point out applications based on IoT and cloud.

In a cloud environment, cloud providers (such as Google, Amazon, and Microsoft) provide computing resources and services, and users can access them just like ordinary applications that are leased or released. At the same time, the cloud provider must manage, store, and ensure the reliability of the resources, and the user only needs to pay according to the usage when the task is performed. In order to satisfy the user's request and obtain the maximum profit, the cloud provider must consider the performance and cost of the cloud system in the resource scheduling process. How to efficiently utilize the resources in cloud computing has always been a core concern [5]. Among them, the literature [6] allocates virtual machines from the perspective of least resource consumption, which saves the cost of computing but relatively prolongs the completion time of the task. We consider how to complete the efficient scheduling of large-scale tasks.

Literature [7] summarizes the division of task scheduling algorithms in cloud environment and divides task scheduling algorithms in cloud environment into two categories: static task scheduling algorithms and dynamic task scheduling algorithms. The unified scheduling after accumulating certain tasks is called static scheduling, and the scheduling arrangement changes with the entry of new tasks, which is called dynamic scheduling. The algorithm proposed in this

paper mainly solves the large-scale static scheduling problem in MCC environment.

The goal of task scheduling is to assign tasks to each virtual machine. Arrange the execution order of tasks, so that they take the shortest time to complete under the premise of satisfying constraints. The task scheduling problem has been shown to be a nondeterministic polynomial (NP) problem [8, 9]. References [10–13] attempt to solve other equally difficult scheduling or assignment problems. So how to optimize the task scheduling decision is a challenging work. To address this challenge, we design an efficient delay-aware task scheduling algorithm (EDTSA), which treats the cloud computing problem as an optimization problem and solves it using a heuristic strategy. The relationship between tasks in multiple stages and virtual machines is modeled as a bipartite graph. The optimal situation of each stage is obtained by network flow algorithm as our initial solution and a local search is used to find an approximate global optimal solution. In addition, we have a new division of the neighborhood, reducing many unnecessary search areas.

The remainder of this article is organized as follows: Section 2 reviews related work. In Section 3, we model the task scheduling problem in cloud computing. In Section 4, the problem transformation is performed and a heuristic algorithm, EDTSA, is proposed to solve the optimization problem. Section 5 presents the experimental results and shows the comparison results of the EDTSA with other algorithms. Section 6 concludes the paper and discusses future directions for this paper.

2. Related Work

There have been many related researches on the task scheduling problem in cloud computing from the model and algorithm. In the optimization task scheduling problem, there are traditional algorithms related to dynamic programming, as well as various heuristic algorithms and algorithms based on reinforcement learning. Xie et al. [14] proposed a cloud scheduling algorithm driven by dynamic basic paths, which computes the dynamic essential path of the prescheduling task nodes based on the actual computation cost and communication cost of task node in the scheduling process. References [15–17] describe how to implement intrusion prevention and optimization on the cloud.

Cui and Xiaoqing [18] proposed a cloud computing workflow task scheduling algorithm based on genetic algorithm, and designed a new genetic crossover and mutation operation to generate new and different offspring, thereby increasing population diversity. Junliang et al. [19] proposes a task scheduling algorithm based on particle swarm ant colony algorithm, which selects the fitness function according to the multicore multitask system model. Dai and Zhang [20] proposed a heuristic task scheduling algorithm for static task scheduling in heterogeneous environments. In the task sorting stage, there are three priorities in the algorithm to select tasks. They all try to optimize the task scheduling problem through heuristic algorithm, which is also the mainstream algorithm for task scheduling problem in cloud computing.

Qi and Zhuo [21] attempted to solve scheduling problems with reinforcement learning and illustrated the drawbacks of traditional algorithms. Chen et al. [22] solved the problem of dynamic unloading in the Internet of Things through in-depth reinforcement learning. Siddique et al. [23] used neural network algorithm to make air quality monitoring system. Park and Yoo [24] has achieved good results in the vehicle scheduling problem with the reinforcement learning method. References [25–27] have achieved good results using deep reinforcement learning in different scenarios in the IoT environment. They all pointed out that reinforcement learning is more forward-looking than other algorithms, not limited to the limitations of local optimal solutions and can have obvious effects in many scenarios.

References [28–31] have proposed resource scheduling or task unloading methods in mobile edge computing (MEC). They also pay attention to the optimization problems in the Internet of Things or cloud environment. Yr and Champa [32] applied IoT data scheduling fuzzy technology with artificial neural network (ANN) to optimize system resources such as memory storage, CPU processing time, and energy consumption. Huang et al. [33] proposed multiqueue approach of energy efficient task scheduling for sensor hubs. Chen et al. [34] optimizes edge caching on IoT services.

In addition, many researchers have proposed new models and frameworks. Wang et al. [35] proposed a binary nonlinear programming model (BNP) model to optimize the task deadline conflict problem. Zhang and Zhou [7] proposed a cloud task scheduling framework based on a two-stage strategy to improve cloud task scheduling and execution results. Ali and Li [36] proposed a cloud test evaluation model based on fuzzy multiattribute decision algorithm.

You et al. [37] decomposed the joint task scheduling problem into multiple subproblems in MEC and formulated an optimization problem to minimize the overall energy consumption of all user UAVs. The branch and bound method and continuous convex approximation technique are used to solve the optimal solution of the subproblem. We also pay attention to the optimal solution of the subproblems, perform global optimization on the combined solutions of the subproblems, and design an effective algorithm that know in advance about the task arrival.

3. System Model

In this paper, we focus on task scheduling in cloud computing environment. We consider a computing environment composed of multiple virtual machines in a cloud computing environment. These virtual machines have computing capacities for processing tasks. Table 1 summarizes notations used in our model.

3.1. Task and Computing Model. There are T tasks, $t_1, t_2, \ldots, t_T$, to be executed. If task t_i is still not processed within the maximum response time after it is generated, the task times out. The timeout amount r_i of task t_i is formulated as follows:

Table 1: Notions and definitions.

Notation	Definition
T	The set of tasks
N	The set of virtual machines
R	The total timeout for all tasks
Q	The queue
L	The length of the queue of pending tasks
g_i	The time when the i-th task was generated
m_i	The maximum response time of the i-th task
p_i	The type of the i-th task
s_i	The time when the i-th task started to be processed
r_i	The amount of timeout for the i-th task
h_j	The number of idle threads of the j-th virtual machine
$V_{\max}(t)$	The maximum number of tasks that can be computed At time t
$V(t)$ rowhead	Choose the number of tasks that can be computed at time t
$L(t)$ rowhead	The length of the queue of pending tasks at time t.
M_{jp} rowhead	The time that the j-th virtual machine needs to process The p-th type of task
P_{jk} rowhead	The next idle moment of the k-th thread in the j-th Virtual machine

$$r_i = \max\{s_i - g_i - m_i, 0\}, \tag{1}$$

where g_i represents the generation time, m_i represents the maximum response time, and s_i represents the task start processing time. The total timeout amount R of all tasks is defined as follows:

$$R = \sum_{i=1}^{T} r_i. \tag{2}$$

We will count all tasks and no tasks will be discarded. Because of the limited computing power of virtual machines, it is impossible for us to have all tasks entering the system at the same time. We put tasks that are not processed at any time into the pending task queue Q, and the queue length is L. We have a total of N virtual machines, each virtual machine has v_j threads. A task is assumed not to be reassigned to another virtual machine during its execution, i.e., the task is non-preemptive, due to reasons such as high migration overheads and penalty. Each thread can only process one current task until the current task is processed by this virtual machine. Assuming that we generate n tasks at the same time at time t and each virtual machine has threads that h_j does not use at this time, the maximum number of tasks that can be calculated at this time $V_{\max}(t)$ should satisfy the following constraints:

$$V_{\max}(t) = \min\left\{L(t) + n, \sum_{j=1}^{N} h_j\right\}. \tag{3}$$

At the same time, we can choose the number of tasks $V(t)$ to be processed at this moment should satisfy the constraint:

$$0 \le V(t) \le V_{\max}(t). \tag{4}$$

During every moment, some computation tasks would be taken from the task queue and then processed by the virtual machine. Meanwhile, some new computation tasks would also arrive. Thus, the queue length evolves as the following equality:

$$L(t+1) = L(t) + n - V(t). \tag{5}$$

Since there are many types of tasks, different virtual machines need different computing times for different types of tasks. M_{jp} represents the time required for the j-th virtual machine to process the p-th type of tasks. If the k-th thread of a virtual machine j-th starts to process a task of the p-th type at time t, the time it can process the next task follows.

$$P_{jk}(n+1) = P_{jk}(n) + M_{jp}. \tag{6}$$

3.2. Optimization Problem. Since this paper focuses on efficiently processing all tasks with limited virtual machines in the cloud computing environment. We want to know which virtual machine should be allocated to handle which task at each moment in order to complete all tasks eventually. The amount of timeout is the smallest, which is obviously an NP-complete problem, and our optimization is to minimize the total timeout amount which is expressed in equation (2). We first sort all tasks by generation time, satisfying the following constraints:

$$g_i \le g_{i+1}, \quad \forall i \in T. \tag{7}$$

If we want to optimize the optimal solution of all problems, we can consider splitting the problem into several subproblems and finding the optimal solution of the subproblems. Being a subproblem is obviously not a particularly efficient way to deal with it, we want to decompose the problem as little as possible and we want the solution to the subproblem to be as good as possible.

4. Algorithm Design

For the task scheduling problem in cloud computing, the time complexity required by conventional algorithms to solve is too large or the solution given in a given time is not excellent. Heuristic algorithms often have better results in solving these NP problems, but for heuristic algorithms, especially for local search and tabu search algorithms, the quality and characteristics of its initial solution are very important.

We know that for all tasks, the optimal solution to solve, it is not necessarily transferred from the optimal solution of its subproblems. Specifically, we solve the optimal solution for all T tasks, where how do we assign $t_i, t_{i+1}, \ldots, t_u$ to which virtual machines these tasks may not be the same as the solution where we only need to solve these tasks, because we not only need to consider this part of the task, but also need to consider other tasks, even the tasks that may arrive in the future. If we only consider the fastest solution for the current task, it may seriously affect the other tasks efficiency,

but if we divide this batch of tasks into as few parts as possible, our subproblem is not to deal with a single task, but to deal with a batch of tasks, we only need to ensure that the efficiency of processing these tasks is very high, and then through the local search to adjust the solution, our algorithm will be able to give a better solution in the given time.

4.1. Problem Transformation. In order to better calculate the task scheduling problem in the cloud computing environment, we will transform the scenario of assigning tasks to virtual machines into a bipartite graph with weights, in which the task t_i is assigned to the virtual machine j for processing, and then it is transformed into a bipartite graph with weights. The task in the graph connects a directed edge to the virtual machine, and the weight is the completion time of the current task processed by the virtual machine. Of course, if we connect each task to all virtual machines and determine whether to select this edge, many unnecessary edges need to be calculated, we are concerned that all tasks are responded by the virtual machine as quickly as possible, so in order to reduce unnecessary calculations, we only compare the tasks currently in the idle queue with tasks that are not currently being processed in the virtual machine and only when the number of idle tasks and the number of currently idle virtual machine threads reaches a certain number or when no tasks will arrive in the future, we will perform a calculation on this batch of tasks. Since we want to divide the task scheduling problem into as few subproblems as possible, we process as many tasks as possible at a time.

We can better calculate and express the relationship between tasks and threads or virtual machines by building a bipartite graph. In the bipartite graph, each idle virtual machine thread has an edge with the current idle task.

4.2. Distributed Online Algorithm for Task Scheduling. Given a bipartite graph $G = (V, E)$, the vertex V is divided into two disjoint subsets (X, Y), where the set of idle threads in the virtual machine belongs to X, and the set of tasks to be processed belongs to Y. In a subgraph F of G, if any two edges in the edge set E of F are not attached to the same vertex, then F is said to be a match. Since we want to work on more tasks at a time, this divides the problem into fewer subproblems. We hope to find a set of maximal matching, which means that under the currently completed matching, the number of matching edges cannot be increased by adding unfinished matching edges. The edge weights selected in this set of maximal matching are $\text{weight}_1, \text{weight}_2, \ldots, \text{weight}_{v(t)_{\max}}$. When we construct a bipartite graph every time, it can be calculated, then the problem can be transformed into how to find the minimum weight matching in all maximal matching of this graph. In other words, we want to select some edges in the graph to satisfy the following formula:

$$\text{cost} = \min\left\{\sum_{e=1}^{V(t)_{\max}} \text{weight}_e\right\}. \tag{8}$$

To find the minimum weight matching of a graph, we can convert it into a network flow algorithm [38] to solve its minimum cost flow. First, when the calculation conditions are met, the effect of building a graph is shown in Figure 1.

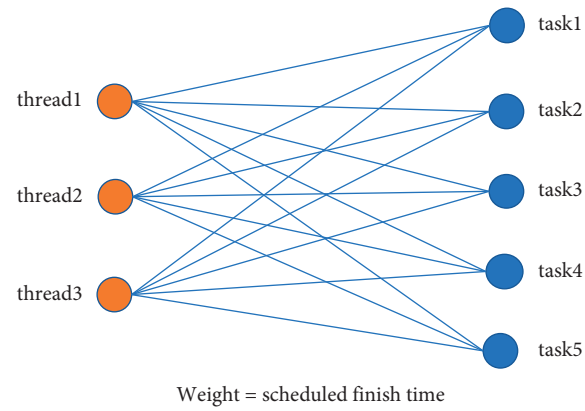

Figure 1: Thread and task relationship diagram.

At this time, the left side represents the number of all idle threads in our virtual machine at the moment and the right side represents the tasks in the pending queue at the moment, as shown in formula (8), we need to find the minimum weight matching of the bipartite graph. We use the Dinic's algorithm to solve the optimal solution of this problem in strong polynomial time. Compared with the Edmonds–Karp algorithm [39], Dinic's algorithm can have great advantages on dense graphs, such as bipartite matching. Dinic's algorithm finds all augmentation paths each time, and then augments it. If there is a ring, there will be countless augmented paths. This problem can be solved with the help of the idea of hierarchical graphs, use the SPFA algorithm to build a hierarchical graph, start from the source point, and expand one layer at a time, each layer is given a number. Then, use the DFS algorithm for each layer to find all paths that can be augmented. The above SPFA algorithm has two functions: (1) establishing a hierarchical graph and (2) judging whether there is an augmentation path.

After each time using SPFA algorithm to build a hierarchical graph, we will use the DFS algorithm to find all augmenting paths corresponding to the current hierarchical graph, in which we can use the current arc optimization. Current arc optimization: for each point, we record which edge it should traverse from, instead of traversing all from the beginning.

We process a batch of tasks each time, so that the processing efficiency of this batch of tasks is the highest, so that when we keep repeating this process until we have completed all tasks, our initial solution is formed, of course, we set different parameters in solving the initial solution, which will cause us to generate different initial solutions. We set A and B to different values, and the initial solution will be different, but no matter what the value is set to, what we seek each time is the optimal solution for a batch of tasks.

4.3. Optimize the Initial Solution. For the initial solution, what we seek is the optimal solution of multiple subproblems and combine them, but this is not necessarily the global optimal solution, so we introduce a heuristic algorithm to

optimize the initial solution. We know that for heuristic algorithms, especially local search algorithms, tabu search algorithms, etc., we are very dependent on the performance of the initial solution. If we are in a poor initial point at the beginning, the final result often not be very ideal, but by using Algorithms 1 and 2, we can construct a better initial solution and we can easily change the initial solution by changing the parameters, which is very beneficial for us to use heuristic algorithms optimized for it.

In this paper, we optimize our solution using a local search algorithm [40], a heuristic algorithm for solving optimization problems. The concept of neighborhood is used to describe the algorithm. The so-called neighborhood is simply a collection of other points near a given point. In the distance space, the neighborhood is generally defined as a circle with a given point as the center. In combination optimization problem, the neighborhood generally defined as the transformation of each node on the given problem domain by the given transformation rule. A collection of nodes on the problem neighborhood, the formula is described as follows:

$$N\colon S \in D \longrightarrow N(S) \in 2^{D}. \tag{9}$$

Then, $N(S)$ is called the neighborhood of N. In our initial solution, we defined the following three neighborhoods:

$$\begin{aligned} &N\colon S \in D \longrightarrow N(S)\colon \operatorname{swap}(t_i, t_u), \quad k_i \neq k_u, \\ &N\colon S \in D \longrightarrow N(S)\colon \text{move } t_i \text{ behind } t_u, \text{ and} \\ &N\colon S \in D \longrightarrow N(S)\colon \operatorname{swap}(t_i, t_u), \quad i < u,\ k_i = k_u. \end{aligned} \tag{10}$$

For these three neighborhoods, the relationship of the two tasks we operate on is described, where the two tasks are separated on different threads, and the exchange needs to satisfy.

$$\begin{aligned} &s_i \geq g_u \text{ and} \\ &s_u \geq g_i. \end{aligned} \tag{11}$$

After the swap, the start time of the tasks processed after the i and u tasks on these two threads needs to change. The second kind of neighborhood is defined as the task i in thread A, which is moved to the task u in thread B for postprocessing. Similarly, the following constraints need to be met:

$$g_i \leq s_u + M_{jp}. \tag{12}$$

The third neighborhood is defined as the exchange of the processing time of tasks i and u in thread A. In other words, if task i is processed in thread A, then task u is processed, and the neighborhood is defined as the thread A processes task u, and then process task i. The prerequisite for the neighborhood to be legal is as follows:

$$g_u \leq s_i. \tag{13}$$

We have defined the neighborhood of a solution. Of course, the neighborhood of a solution is very large. If we search in such a large neighborhood, it may take a very long time to optimize the initial solution. We found that although many neighborhoods meet the definition, it is impossible to make our solution better. For example, when the generation time of task i and the generation time of task u are far away, if we still have to exchange these two tasks, at least one of task i and task u incurs a large number of timeouts, so the task we operate each time needs to satisfy the task start time in the original solution that does not exceed, that is, it conforms to the following formula:

$$\left|s_i - s_u\right| \leq. \tag{14}$$

Moreover, we set up two tabu tables to prevent our search from falling into an infinite loop. One of them is a one-dimensional tabu table, which records when we operate on a task and the other is a two-dimensional tabu table, which records which task we operate on each time and on which thread it is currently being processed. We have set up two tabu periods, which correspond to two tabu tables. We need to satisfy the following equation when performing neighborhood transformation:

$$\begin{aligned} &\text{time} - \text{operate}_i > tt_1 \text{ and} \\ &\text{time} - \text{operate}_{i,j} > tt_2. \end{aligned} \tag{15}$$

To prevent getting stuck in a local optimum, we accept this solution with a certain probability if the quality of the neighborhood solution degrades within our allowable range (Algorithm 3) [40].

We represent the relationship between tasks and virtual machines or threads by constructing a bipartite graph and construct an excellent initial solution through the minimum cost flow algorithm to ensure the quality of the initial solution. Because we only need to change the parameters, we can generate different initial solutions , so we can generate k groups of initial solutions with different parameters under the condition of time permitting, and then use the local search algorithm to find the optimal solution of each groups. We compare the k groups of initial solutions and choose the best one as our final solution.

5. Experiments

In this section, we conduct experiments to evaluate our task scheduling method in various performance metrics and analyze the results.

5.1. Experiment Design. We conduct experiments to evaluate the efficiency of the scheme given by our final solution. In the system, we have 100 virtual machines and 7000 tasks arrived in 1440 seconds. We tested the efficiency of the EDTSA under different conditions in our experiments. We can adjust the efficiency of the virtual machines, the number of threads each virtual machine has to dynamically change the difficulty of task scheduling problems.

In this paper, we designed different scenarios to test the effect of EDTSA, and selected two benchmark algorithms.

(1) The MIN-MIN algorithm [41], which is similar to our algorithm, is used to process the optimal solution

Input: All tasks sorted by task spawn time
Output: Bipartite graph between tasks and threads
(1) **for** All task $t_i \in T$ **do**
(2) **if** $L \geq A, \sum_{j=1}^{N} h_j \geq B$ **then**
(3) **for** all $q \in Q$ **do**
(4) **for** all $j \in N$ **do**
(5) **if** $h_j > 0$ **then**
(6) add edge $q \longrightarrow h$
(7) weight = $P_{jk} + M_{jp}$
(8) **end if**
(9) **end for**
(10) **end for**
(11) calc $V_{\max}(t)$ task
(12) erase calc task in Q
(13) And the length of L also changes accordingly
(14) **else**
(15) $t_i \longrightarrow Q$
(16) $L++$
(17) **end if**
(18) **end for**

Algorithm 1: Construct a bipartite graph.

Input: Bipartite graph between tasks and threads
Output: initial solution
(1) **while** we can use SPFA algorithm to find augmenting path **do**
(2) Augmentation with DFS algorithm
(3) Accumulate max flow, min fee
(4) Reduce flow on the forward side, add flow on the reverse side
(5) **end while**
(6) **for** all edge E_e in this graph **do**
(7) **if** E_e in least cost side **then**
(8) cost += w_e
(9) pair += $\{k, i\}$
(10) **end if**
(11) **end for**
(12) Return initial solution and cost

Algorithm 2: Dinic's algorithm to find the optimal solution.

Input: initial solution
Output: optimized solution
(1) **while** time < time limit **do**
(2) find neighborhood solutions
(3) **if** neighborhood transform operation ‘ tt **then**
(4) continue
(5) **end if**
(6) **if** Delta cost ≥ 0 **then**
(7) move the current solution to the neighborhood solution
(8) **else if** range < Delta cost < 0 **then**
(9) move the current solution to the neighborhood solution
(10) **end if**
(11) **end while**
(12) Return solution and cost

Algorithm 3: Optimize the initial solution.

of the subproblems and finally merge, but the difference is that the MIN-MIN algorithm only focuses on the optimal solution of a single task, and each time when a task is generated, the algorithm will assign the task to the virtual machine that completes first. Although the current task can be completed the fastest, it ignores the impact on other tasks.

(2) Random algorithm, although the random algorithm is not competitive with the current mainstream algorithms, we can reflect the processing difficulty of a set of tasks through the random algorithm. If the random algorithm can handle a set of tasks well, then it is proved that this set of tasks is very easy.

5.2. Comparison Experiment. We evaluate the algorithm from two aspects: the amount of task timeout and the proportion of overtime tasks. The smaller the task timeout amount and the smaller the proportion of overtime tasks, the better the performance of the algorithm.

We compare the performance of our algorithm with other algorithms in different scenarios. Figure 2 show the change in the amount of timeouts and the proportion of timed out tasks as the number of tasks to be processed increases, where the unit of timeout amount is seconds. Although the random algorithm is not competitive, it can clearly reflect the difficulty of solving this group of tasks. We can see that both our algorithm and the MIN-MIN algorithm can complete the task on time when the task volume is small at the beginning. As the amount of tasks increases, the MIN-MIN algorithm begins to have tasks that have timed out, and then when about 500 tasks are generated, our algorithm will have task timeouts. As can be seen from Figure 3, when the amount of tasks increases, the performance gap between different algorithms is very obvious. The random algorithm performed very poorly in this set of experiments, which also shows that this set of tasks is difficult to handle.

When the number of tasks arriving is certain and the efficiency of the virtual machine is certain, we adjust the number of threads of each virtual machine. Figures 4 and 5 show the changes in the task timeout amount and task timeout ratio in this scenario. Because the random algorithm is only to reflect the difficulty of solving a set of tasks, we no longer consider the random algorithm in other scenarios. We set the number of threads of each virtual machine to 1, 2, 4, and 8 respectively. It can be seen that in different environments, the performance of our algorithm is better than that of the MIN-MIN algorithm, and as the number of threads increases, the processing tasks speed has also accelerated.

When the number of tasks arriving is certain and the number of threads per virtual machine is certain, we adjust the efficiency of the virtual machine to adjust the difficulty of the problem. Figures 6 and 7 show the efficiency and comparison of our algorithm in this scenario. We adjust the efficiency of the virtual machine to 1/2, 1/3, 1/4, and 1/5 of the normal efficiency, and the data show that our algorithm is still better than the MIN-MIN algorithm in the case of very low virtual machine efficiency.

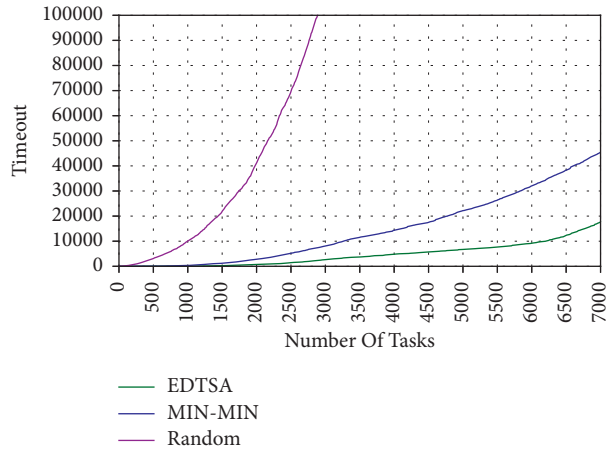

Figure 2: As the number of tasks increases, the amount of timeout changes.

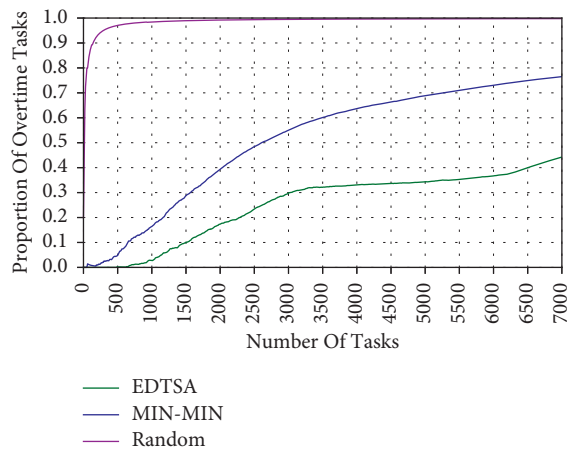

Figure 3: As the number of tasks increases, the proportion of overtime tasks changes.

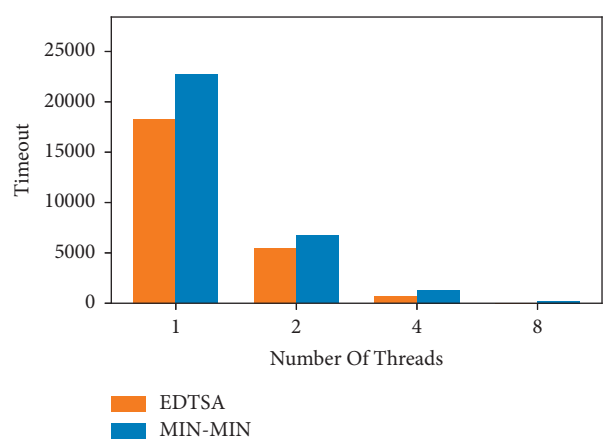

Figure 4: The amount of timeout changes with different threads of the virtual machine.

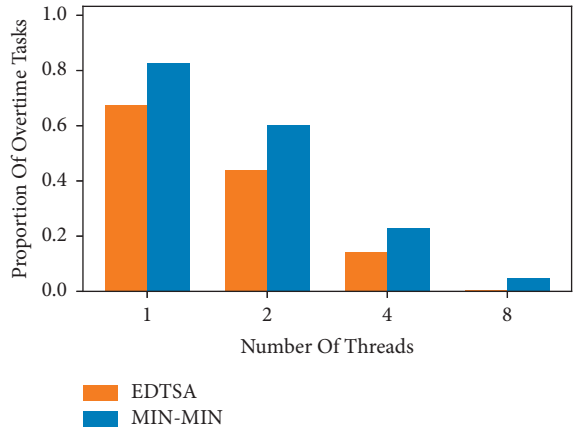

Figure 5: The proportion of timeout tasks changes with different threads of the virtual machine.

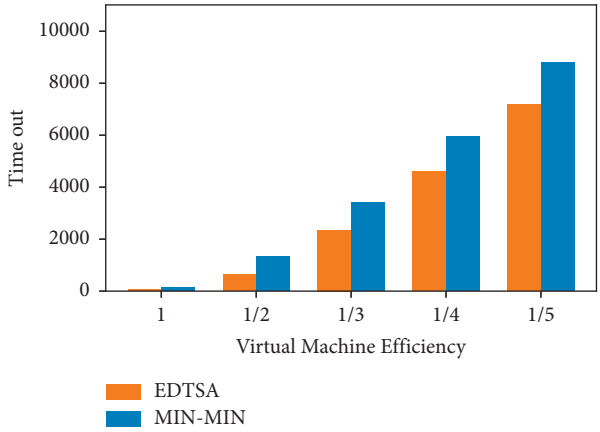

Figure 6: The amount of timeout changes with different virtual machine efficiency.

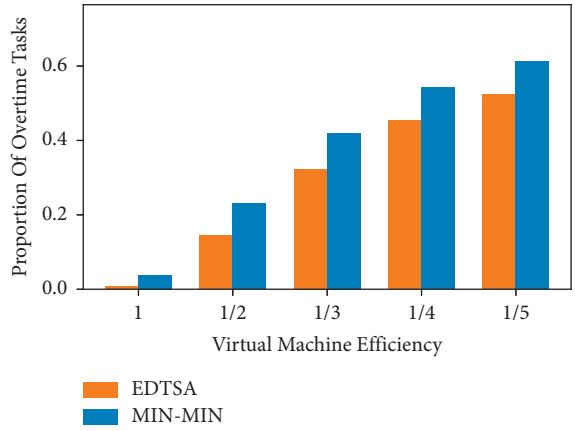

Figure 7: Changes in the proportion of timed out tasks with different virtual machine efficiencies.

According to the experimental data, it can be seen that our algorithm is better than the comparison algorithm in various scenarios and as the task difficulty increases, the performance gap between our algorithm and the comparison algorithm will be larger, which indicates that the proposed algorithms are efficient and practical.

6. Conclusions

In this article, we study the task scheduling problem in cloud computing. We model the relationship between tasks and virtual machines as a bipartite graph and propose a heuristic algorithm EDTSA to solve the optimal solution. The experimental results show that the EDTSA has high solution efficiency in various environments. For our future work, we will employ deep reinforcement learning techniques to solve the task scheduling problem in cloud computing.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 61902029), R&D Program of Beijing Municipal Education Commission (No. KM202011232015), Project for Acceleration of University Classification Development (No 5112211036, 5112211037, 5112211038), and BISTU College Students Innovation and Entrepreneurship Training Program (No 5112210832).

References

[1] J. Huang, Z. Tong, and Z. Feng, "Geographical poi recommendation for internet of things: a federated learning approach using matrix factorization," *International Journal of Communication Systems*, p. e5161, 2022.

[2] I. Ullah, B. Raza, S. Ali, I. A. Abbasi, S. Baseer, and A. Irshad, "Software defined network enabled fog-to-things hybrid deep learning driven cyber threat detection system," *Security and Communication Networks*, vol. 2021, Article ID 6136670, 15 pages, 2021.

[3] S. Ali, S. Baseer, I. A. Abbasi, and B. Alouffi, "Analyzing the interactions among factors affecting cloud adoption for software testing: a two-stage ism-ann approach," *Soft Computing*, pp. 1–29, 2022.

[4] L. Qi, W. Lin, X. Zhang, W. Dou, X. Xu, and J. Chen, "A correlation graph based approach for personalized and compatible web apis recommendation in mobile app development," *IEEE Transactions on Knowledge and Data Engineering*, p. 1, 2022.

[5] M. Armbrust, A. Fox, R. Griffith et al., "Above the clouds: a berkeley view of cloud computing," EECS Department, University of California, Technical Report UCB/EECS-2009-28, 2009.

[6] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Proceedings of the 2011 Third International Conference on Communications*

and mobile Computing, pp. 122–125, IEEE, Qingdao, China, June 2011.

[7] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, 2018.

[8] S. Tayal, "Tasks scheduling optimization for the cloud computing systems," *International journal of advanced engineering sciences and technologies*, vol. 5, no. 2, pp. 111–115, 2011.

[9] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[10] S. Fu, J. Gao, and L. Zhao, "Integrated resource management for terrestrial-satellite systems," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3256–3266, 2020.

[11] Q. Li, L. Zhao, J. Gao, H. Liang, L. Zhao, and X. Tang, "Smdp-based coordinated virtual machine allocations in cloud-fog computing systems," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1977–1988, 2018.

[12] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4925–4934, 2020.

[13] H. A. Shah and L. Zhao, "Multiagent deep-reinforcement-learning-based virtual resource allocation through network function virtualization in internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3410–3421, 2021.

[14] Z. Xie, X. Shao, and Y. Xin, "A scheduling algorithm for cloud computing system based on the driver of dynamic essential path," *PLoS One*, vol. 11, no. 8, Article ID e0159932, 2016.

[15] L. Qi, Y. Yang, X. Zhou, W. Rafique, and J. Ma, "Fast anomaly identification based on multiaspect data streams for intelligent intrusion detection toward secure industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6503–6511, 2022.

[16] W. Wei, R. Yang, H. Gu, W. Zhao, C. Chen, and S. Wan, "Multi-objective Optimization for Resource Allocation in Vehicular Cloud Computing Networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2021.

[17] L. Qi, C. Hu, X. Zhang et al., "Privacy-aware data fusion and prediction with spatial-temporal context for smart city industrial environment," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4159–4167, 2021.

[18] Y. Cui and Z. Xiaoqing, "Workflow tasks scheduling optimization based on genetic algorithm in clouds," in *Proceedings of the 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pp. 6–10, IEEE, Chengdu, China, April 2018.

[19] L. Junliang, H. Wei, S. Huan, L. Yaxin, and L. Jing, "Particle swarm algorithm based task scheduling for many-core systems," in *Proceedings of the 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1860–1864, IEEE, Chengdu, China, June 2018.

[20] Y. Dai and X. Zhang, "A synthesized heuristic task scheduling algorithm," *The Scientific World Journal*, vol. 2014, Article ID 465702, 9 pages, 2014.

[21] F Qi and L Zhuo, "Deep reinforcement learning based task scheduling in edge computing networks," in *Proceedings of the 2020 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 835–840, IEEE, Chongqing, China, November 2020.

[22] Y. Chen, W. Gu, and K. Li, "Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning," *International Journal of Communication Systems*, Article ID e5154, 2022.

[23] A. B. Siddique, R. Kazmi, H. U. Khan, S. Ali, A. Samad, and G. Javaid, "An intelligent and secure air quality monitoring system using neural network algorithm and blockchain," *IETE Journal of Research*, pp. 1–14, 2022.

[24] S. Park and Y. Yoo, "Real-time scheduling using reinforcement learning technique for the connected vehicles," in *Proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, IEEE, Porto, Portugal, June 2018.

[25] J. Xu, D. Li, W. Gu, and Y. Chen, "Uav-assisted Task Offloading for Iot in Smart Buildings and Environment via Deep Reinforcement Learning," *Building and Environment*, Article ID 109218, 2022.

[26] C. Chen, Y. Zhang, Z. Wang, S. Wan, and Q. Pei, "Distributed computation offloading method based on deep reinforcement learning in icv," *Applied Soft Computing*, vol. 103, Article ID 107108, 2021.

[27] S. Liu, J. Yu, X. Deng, and S. Wan, "Fedcpf: an efficient-communication federated learning approach for vehicular edge computing in 6g communication networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 1616–1629, 2022.

[28] Y. Chen, F. Zhao, Y. Lu, and X. Chen, "Dynamic task offloading for mobile edge computing with hybrid energy supply," *Tsinghua Science and Technology*, vol. 10, 2021.

[29] Y. Lu, X. Chen, Y. Zhang, and Y. Chen, "Cost-efficient Resources Scheduling for mobile Edge Computing in Ultra-dense Networks," *IEEE Transactions on Network and Service Management*, p. 1, 2022.

[30] Y. Chen, F. Zhao, X. Chen, and Y. Wu, "Efficient multi-vehicle task offloading for mobile edge computing in 6g networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 4584–4595, 2022.

[31] J. Huang, B. Lv, Y. Chen, Y. Shen, and X. Shen, "Dynamic admission control and resource allocation for mobile edge computing enabled small cell network," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, p. 1964, 2022.

[32] S. K. Yr and H. N. Champa, "A Fuzzy Based Scheduling Approach for Efficient Sensing in Iot-Cloud," in *Proceedings of the 2021 12th International Conference On Computing Communication And Networking Technologies (ICCCNT)*, pp. 1–5, IEEE, Kharagpur, India, July, 2021.

[33] J. Huang, C. Zhang, and J. Zhang, "A multi-queue approach of energy efficient task scheduling for sensor hubs," *Chinese Journal of Electronics*, vol. 29, no. 2, pp. 242–247, 2020.

[34] Y. Chen, H. Xing, Z. Ma, and X Chen, "Cost-efficient Edge Caching for Noma-Enabled Iot Services," *China Communications*, 2022.

[35] B. Wang, Y. Song, C. Wang, W. Huang, and X. Qin, "A study on heuristic task scheduling optimizing task deadline violations in heterogeneous computational environments," *IEEE Access*, vol. 8, Article ID 205635, 2020.

[36] S. Ali and H. Li, "Moving software testing to the cloud: an adoption assessment model based on fuzzy multi-attribute decision making algorithm," in *Proceedings of the 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, pp. 382–386, IEEE, Tokyo, Japan, April 2019.

[37] W. You, C. Dong, Q. Wu, Y. Qu, Y. Wu, and R. He, "Joint task scheduling, resource allocation, and uav trajectory under clustering for fanets," *China Communications*, vol. 19, no. 1, pp. 104–118, 2022.

[38] A. V. Goldberg, É. Tardos, and R. Tarjan, "Network Flow Algorithm," Cornell University Operations Research and Industrial Engineering, Ithaca, NY, US, Tech. Rep, 1989.
[39] N. Zadeh, "Theoretical efficiency of the edmonds-karp algorithm for computing maximal flows," *Journal of the ACM*, vol. 19, no. 1, pp. 184–192, 1972.
[40] E. Aarts, E. H. Aarts, and J. K. Lenstra, *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, New Jersey, US, 2003.
[41] X. He, X. Sun, and G. von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *Journal of Computer Science and Technology*, vol. 18, no. 4, pp. 442–451, 2003.