Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 3660882, 11 pages
https://doi.org/10.1155/2023/3660882

*Research Article*

# Reinforcement Learning-Based Intelligent Task Scheduling for Large-Scale IoT Systems

**Chenghou Jin,[1] Yusen Han,[1] Zhuo Deng,[1] Ying Chen,[1] Chengxia Liu,[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 13 September 2022; Revised 17 October 2022; Accepted 31 January 2023; Published 23 February 2023

Academic Editor: Sotirios K. Goudos



The new-generation of Internet of Things (NG-IoT) brings a wide range of challenging problems. At the same time, cloud computing technology is an important foundation for the development of the IoT. In this article, we focus on the task scheduling problem in IoT systems in cloud computing environment. Our goal is to minimize the task runtime. It is well known that the problem of the task scheduling has been a challenging problem. In the last decade, despite being theoretically hard problem, researchers design lots of state-of-the-art algorithms for solving this problem. In our work, we propose a novel efficient reinforcement learning (RL) algorithm to solve the task scheduling problem in IoT systems (EATS), which combines combinatorial optimization to make our proposed algorithm have stable lower bounds. We process a batch of tasks at a time, make decisions on task selection through reinforcement learning, and solve them further through combinatorial optimization methods. The results of the experiments show that our proposed algorithm has outstanding performance in different environments.

## 1. Introduction

The application of NG-IoT in many fields is becoming more and more popular. However, enormous number of Internet of Things (IoT) systems generates a large capacity of data, and how to efficiently process these data is becoming more and more important. The task scheduling problem in IoT systems refers to allocating tasks generated in IoT systems to virtual machines, so that the total time required is the least. As we all know, the problem of task scheduling has always been an NP problem. How to deal with this problem more efficiently has always been a challenging work. Artificial intelligence (AI) algorithm plays an important role in various IoT problems. Recently, Zhou et al. [1] proposed an accelerating artificial intelligence method for IoT (AAIoT) for the first time. Christou et al. [2] introduce an industrial IoT model based on machine learning (ML) and discuss its explainable.

We know that some problems in IoT are often difficult. Of course, there are many researchers who treat these problems as optimization problems and solve them well. Fu et al. [3] proposes an optimization method for resource management in terrestrial satellite systems. Liu and Zhang [4] proposed a joint optimization algorithm based on the Lagrange dual optimization with the aim of maximizing the transmission rate of the IoT. With the development of the IoT and cloud computing, these issues are becoming increasingly important.

IoT resources are very limited, and how to efficiently use IoT resources has always been an important problem. We study the task scheduling problem in IoT systems, which is a classical NP problem. For this class of problems, many heuristic algorithms and traditional algorithms are widely used. Most of the time, they can only solve small-scale problems. However, for large-scale problems, these algorithms do not perform well, and there are many researchers who solve this class of problems by using deep reinforcement learning. The task scheduling problem is naturally in line with the Markov decision-making process, and Li et al. [5] solve the task division and scheduling problem through deep reinforcement learning technology, and the proposed method has excellent performance. Chen et al. [6] proposed a deep

reinforcement learning- (DRL-) based approach for dynamic task unloading on MEC. They both point out the flaws of traditional algorithms and solve the problem efficiently by reinforcement learning algorithms.

Zhang and Zhou [7] summarize the divided scheduling algorithms in different environments and then divide them two types: static task scheduling and dynamic task scheduling. Uniform scheduling after accumulating a batch number of tasks is known as a static scheduling algorithm, while scheduling arrangements that change as new tasks come in are called dynamic scheduling algorithms. In this article, we concentrate on the problem of scheduling tasks in large-scale static IoT systems. To better solve this problem, we propose an efficient reinforcement learning algorithm for solving this task scheduling problem (EATS) in IoT systems. We use reinforcement learning to guide which tasks are computed each time and solve the optimal solution for each selected task by a combinatorial optimization algorithm. Our proposed algorithm has a stable lower bound and can excel in solving task scheduling problems in large-scale IoT systems.

The algorithm proposed in this article is based on reinforcement learning and combines a combinatorial optimization algorithm to solve task scheduling in large-scale IoT systems. This strategy of combining reinforcement learning with other algorithms often has outstanding results. The use of some algorithm-assisted reinforcement learning algorithms is also currently a popular approach to ensure stable solution lower bounds on difficult problems [8]. With the rapid increases in the quantity of IoT systems, the scale of the task scheduling problem will be enormous. Therefore, it is crucial to design an efficient scheduling algorithm to solve this problem.

This article's remaining sections are organized as follows. In Section 2, we model the problem of task scheduling in IoT systems and show problem formula. In Section 3, we propose a novel reinforcement learning algorithm, EATS, which combines a combinatorial optimization algorithm to excel in solving task scheduling problems in large-scale IoT systems, and we give a proof of a lower bound on the algorithm. In Section 4, we compare EATS with other algorithms in different settings. Section 5 reviews related work. Section 6 summarizes our work and describes our future work.

## 2. System Model and Problem Formulation

In this article, we study the problem of task scheduling in IoT systems in a cloud computing environment. Table 1 shows our notions and definitions. To better solve this problem, the following will describe this problem in more detail and show our system model.

*2.1. Task Model.* IoT system generates tasks every second, there will be $N$ tasks arriving in $T$ seconds, and these tasks will be handled by the virtual machine. When the amount of tasks is large, we will select $n$ tasks from the currently unprocessed task set $S$, for priority processing. The size of the set $S$ is denoted as $S_n$. The task we choose to process each time in $S$ is denoted by $A_S^n$. Each task is either processed as soon as it is generated or queued waiting to be processed. In our model, all tasks in the system are processed and no tasks are discarded.

Each task $i$ has its time $t_i$ when it is generated in IoT system. The type of the $i$-th task is $p_i$, its maximum response time is $m_i$, and its time to be processed by the virtual machine is $w_i$. The time-out amount $\tau_i$ of task $k_i$ is formulated as

$$\tau_i = \max\{w_i - t_i - m_i, 0\}. \tag{1}$$

In this article, our goal is to minimize the sum of time-outs for all tasks in the systems. The sum of the time-outs of all tasks generated in the IoT system is $R$. The smaller the $R$, the more efficient the designed algorithm. $R$ is defined as

$$R = \sum_{i=1}^{N} \tau_i. \tag{2}$$

Tasks have multiple attributes. In this paper, we concentrate on the static task scheduling algorithms in IoT systems where we know in advance when a task is generated.

*2.2. Virtual Machine Model.* In our system model, all tasks are processed by virtual machines. We have $M$ virtual machines in total. Each virtual machine $j$ has $h_j$ threads. At the same time, each thread in the virtual machine can only process one task. $f_j$ is the number of idle threads on virtual machine $j$. Since there are many types of task, the time required of each virtual machine $j$ to calculate the $p$-th task is $u_{jp}$. If the $e$-th idle thread in the virtual machine $j$ starts to calculate the $p$-th type of task at moment $t$, it will be idle at the next time; $P_{je}$ is formulated as

$$P_{je}(n+1) = P_{je}(n) + u_{jp}. \tag{3}$$

For ease of calculation, all virtual machines in this paper have the same number of threads and the same efficiency, but different virtual machines handle different kinds of problems with different efficiency. Tasks cannot be assigned to working threads in the virtual machine; only currently idle threads can process tasks.

*2.3. Problem Formulation.* An efficient scheduling algorithm will result in less total time-out $R$. We process a batch of tasks at a time, rather than focusing on one task at a time. We process $n$ tasks at a time, and the remaining tasks are put into the queue $Q$. The set $S$ is the tasks that have not been processed at present, and these tasks will be put into the queue $Q$ at the first time. $L$ is the length of the queue. $n_{\max}(t)$ is the maximum amount of tasks we can handle at time $t$. $\varepsilon$ represents that amount of tasks generated in IoT systems at time $t$. At this moment, $n_{\max}(t)$ is the maximum amount of tasks we can handle, and it should satisfy the

Table 1: Notions and definitions.

| Notation | Definition |
|---|---|
| $N$ | The set of tasks. |
| $M$ | The set of virtual machines. |
| $S$ | The set of tasks that have not been processed. |
| $n$ | The number of tasks each time we choose to handle. |
| $A_S^n$ | The set of tasks selected from set $S$ each time we choose to handle. |
| $t_i$ | The moment when the $i$-th task was generated in IoT system. |
| $m_i$ | The $i$-th task's maximum response time. |
| $p_i$ | The type of the $i$-th task. |
| $w_i$ | The moment when the $i$-th task started to be calculate. |
| $\tau_i$ | The $i$-th task's time-out amount. |
| $R$ | The total time-out for all task. |
| $h_j$ | The number of threads that virtual machine $j$ has equipped. |
| $f_j$ | The number of idle threads of the $j$-th virtual machine. |
| $u_{jp}$ | The time that the $j$-th virtual machine needs to process the $p$-th type of task. |
| $P_{je}$ | The next idle time of the $e$-th thread in the $j$-th virtual machine. |
| $Q$ | The queue of pending tasks. |
| $L$ | The length of the $Q$. |
| $L(t)$ | The length of the $Q$ at time $t$. |
| $n_{\max}(t)$ | The maximum number of task that can be calculate at time $t$. |

following constraints:

$$n_{\max}(t) = \min\left\{L(t) + \varepsilon, \sum_{j=1}^{M} f_j\right\}. \tag{4}$$

We will select $n$ tasks $A_S^n$ to process at this moment; $n$ should satisfy the following constraint:

$$0 \le n \le n_{\max}(t). \tag{5}$$

There will be $\varepsilon$ tasks generated at each moment, and we will process $n$ tasks; the queue length and the set $S$ will change. Thus, the queue length changes as the following equality:

$$L(t+1) = L(t) + \varepsilon - n. \tag{6}$$

In our system model, all types of tasks are placed in a queue, which simplifies the model without affecting the calculation results.

If we need to process a large number of tasks, one queue may not be able to hold enough. In this case, we will create a new queue and process the tasks in the new queue after all the tasks in the previous queue have been processed.

The new task generated by the IoT system at each moment first enters the waiting queue $Q$ and updates it to the set $S$ at the same time.

Since $n_{\max}(t)$ is sometimes very large, we can choose $n_{\max}(t) + 1$ number as the size of $n$ each time, and there are $2^{A_S^n}$ possibilities for $A_S^n$. So how to choose $n$ and $A_S^n$ is worth considering.

In our preliminary experiments, we found that the choice of which batch of tasks to compute each time had a huge impact on the results. Of course, choosing which batch of tasks to compute each time is also a challenging problem, and in the next section, we will propose a novel reinforcement learning algorithm that learns from failure to better guide task selection.

## 3. Algorithm Design

The task scheduling problem in IoT systems is a challenging problem. As an NP problem, traditional algorithms cannot solve it in polynomial time. Reinforcement learning is a good way to solve this kind of problems, but it takes a long time to train to give a perfect solution. We propose a reinforcement learning algorithm with a lower bound, which can also give an excellent solution after less training.

We define a subproblem as scheduling a batch of tasks. We design a novel reinforcement learning algorithm that combines combinatorial optimization algorithms. Reinforcement learning selects the subproblems we process each time, and combinatorial optimization algorithms can obtain optimal solutions to the subproblems.

*3.1. Problem Transformation.* We match the task $i$ generated in the IoT system with the thread $h$ in the virtual machine, indicating that the task $i$ is processed by the thread $h$. Since we do not need to match working threads with tasks, we only consider idle threads $f$ in thread $h$. Algorithm 1 shows our matching process.

In order to better show the relationship of tasks and idle threads, we model it as a bipartite graph [9], as shown in Figure 1.

In a bipartite graph with weights $G = (V, E)$, the set on left is the set of currently selected batch of tasks, and the set on the right is the set of currently idle thread in virtual machine. An edge $\{v1, v2\}$ in $G$ refers to the assignment of task $v1$ to thread $v2$ for processing, and the weight of this edge is the completion moment after task $v1$ is assigned to thread $v2$ for processing [10].

The weighted bipartite graph can not only clearly represent the relationship between tasks and idle threads but also can be calculated more conveniently.

In Algorithm 1, we decide the set $A_S^n$ one at a time by the reinforcement learning algorithm (Algorithm 2), and we connect the tasks in it to all currently idle threads in the virtual machine (line 3 and line 4) with an edge whose weight is the moment when the task is completed (line5). Finally, we need to modify the corresponding set (line9-line11). Each time the above process is executed, a batch of tasks is processed. As tasks arrive, we keep repeating this process until all tasks have been processed.

*3.2. Novel Reinforcement Learning Algorithm.* There are many possibilities for the choice of $A_S^n$, and we cannot try all of them in polynomial time. We guide the selection of $A_S^n$ through a reinforcement learning algorithm.

$Q$-learning [11] is an algorithm of the value-based in reinforcement learning algorithms. Among them, $Q$ is $Q(s, a)$, which is the expectation that in the state of $s(s \in S)$ at a certain moment, taking action $a(a \in A)$ can obtain the expectation of income. The method of this algorithm is to construct a $Q$-table that combines states and actions, and the stored $Q$ value represents the maximum benefit that we can obtain by selecting this action in this state. The environment will receive the corresponding reward $r$ according to the action of the agent, and this algorithm will choose the optimal decision according to the $Q$ value.

Agent, environment ($E$), reward ($R$), and action ($A$) can abstract the problem into an MDP process, and each completed task sequence is regarded as a state $S_t$; $\pi(a, s)$ is to take action a policy in state $s$. $P_{ss'}^a$ is the probability of selecting action $a$ in state $s$ to transition to the next state $s'$. $R(s'|s, a)$ represents the reward of taking action a and transferring to $s'$ in state $s$. Our goal is to find a strategy that can process all tasks and obtain the maximum reward. Discount factor is recorded as $\gamma$. Horizon is recorded as $H$. Our goal is the following formula:

$$\text{Goal} : \max_\pi E\left[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1})|\pi\right]. \tag{7}$$

**Input:** The task set $A_S^n$ selected each time
**Output:** Bipartite graph of tasks and idle threads
1: **for** *All* tasks $i \in A_S^n$ **do**
2: **for** *all* $j \in M$ **do**
3: **if** $f_j > 0$ **then**
4: add edge $f_j \longrightarrow i$
5: $weight = P_{jf} + u_{jp}$
6: **end if**
7: **end for**
8: **end for**
9: erase $A_S^n$ in $Q$.
10: erase $A_S^n$ in $S$
11: The length of $L$ also modifies accordingly

Algorithm 1: Match tasks with idle threads.

The $Q$-learning algorithm has advantages in offline learning; it uses the bellman equation to deal with the decision problem of the MDP process. The state value function $V$ is used to evaluate whether the current state is excellent. The value of each state is not only determined by the current state but also related of the subsequent state. This makes actions in the current state more forward-looking. The $Q$ -learning algorithm is used to deal with discrete problems; it is naturally suitable for dealing with scheduling problems. For scheduling problems, the action space corresponding to each state is very large, and we will discuss how to solve this problem later.

We briefly introduced the $Q$-learning algorithm, which we combined with the combinatorial optimization algorithm. When an $A_S^n$ to be calculated is given, we model it and the idle threads in the virtual machine as a bipartite graph $G = (V, E)$, where $V$ denotes the number of vertices and $E$ denotes the number of edges. The network flow algorithm Hopcroft-Karp [12] can calculate the cost of $A_S^n$, in which the time complexity is $O(E\sqrt{V})$. The cost is the following formula:

$$\text{cost} = \min\left\{\sum_{e=1}^{n} \text{weight}_e\right\}. \tag{8}$$

We can determine $A_S^n$ through reinforcement learning and find a set of solutions that minimize the total cost of $A_S^n$ through the Hopcroft-Karp (HK) algorithm. In our solution, we model the task and idle virtual machine as a bipartite graph with weights and further transform the problem into a minimum cost maximum flow, which is solved by a combinatorial optimization algorithm.

However, the optimization goal of HK algorithm is formula (8), which is not completely equivalent to the optimization goal of minimizing the time-out amount. We need to further guide this process by reinforcement learning. More specifically, our reward function will be related to the average time-out amount solved by HK algorithm each time. By combining with combinatorial optimization algorithms, our algorithm can have a stable lower bound. In more detail, our algorithm outperforms a simple greedy algorithm

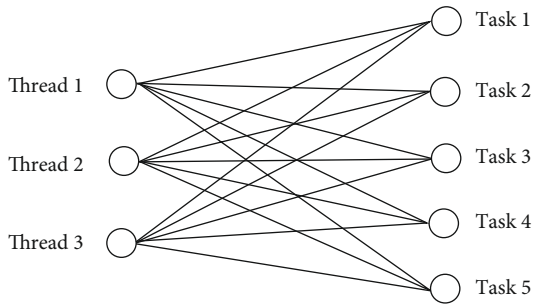


Figure 1: The relationship between idle threads and tasks.

**Input:** All tasks
**Output:** Final solution and timeout
1: Episode = 0;
2: **while** Episode <*bound***do**
3: **for***alltasks***do**
4: **if***random*.rand () < *greedy***then**
5: $next - action = \max\{Q - table[state]\}$
6: **else**
7: $next - action = random.select \in S$
8: **end if**
9: $expected_{\max} = \max\{Q - table[state|next - action]\}$
10: Build a bipartite graph of $next - action$;
11: Calculate cost by HK algorithm;
12: $reward = (tt - cost) + \gamma * expected_{\max}$
13: $Q - table[state|next - action]$E$reward * rate$
14: **if**$\sum_{k=1}^{K} n_k = N$**then**
15: break;
16: **end if**
17: **end for**
18: Episode = Episode + 1;
19: **end while**
20: Return final solution and cost

Algorithm 2: RL guides task selection.

regardless of the tasks chosen by reinforcement learning. Because we always solve for the minimum cost of a batch of tasks, greedy algorithms always only consider a single task that is currently coming.

*Proof.* Let $\text{weight}_k$ be the cost of selecting $A_S^n$ for the $k$-th time; all tasks are completed when the $K$-th time is completed, and let $g_i$ be the cost of processing of the $i$-th task by the greedy algorithm with the shortest task first.

$$N = \sum_{k=1}^{K} n_k, \tag{9}$$

$$\sum_{i=1}^{n_k} g_i \geq \text{weight}_k, \tag{10}$$

$$\cos t_1 = \sum_{i=1}^{N} g_i \geq \sum_{k=1}^{K} \text{weight}_k = \cos t_2. \tag{11}$$

In the above formula, $\cos t_1$ represents the cost of the greedy algorithm that does the shortest task first to complete all tasks, and $\cos t_2$ represents the cost of each time the reinforcement learning algorithm selects an $A_S^n$ to complete all tasks. □

We next discuss how to guide each chosen task through reinforcement learning. Algorithm 2 demonstrates this process.

We use the $Q$-table to judge the quality of each action and guide the choice of the next action $A_S^n$. We set the parameter greedy, which represents how much probability we have to choose the best action in the past (line 4 and line 5). We set it to 0.2; the higher the value, the more likely it is to fall into a local optimum. $\gamma$ and rate are our learning parameters, which represent how much we also receive rewards (line 11 and line 12). It is worth noting that we want the total time-out for the task to be smaller, so when making the reward, we set each reward to a constant tt minus the average time-out cost, so that reinforcement learning can guide us to options with less time-out.

We combine the combinatorial optimization algorithm in lines 9 and 10 of Algorithm 2, which gives us a stable lower bound. In the past, reinforcement learning algorithms were more inclined to choose which virtual machine to assign $\text{task}_i$, but we choose which batch of tasks to calculate each time through reinforcement learning and then use the combinatorial optimization algorithm to calculate the minimum cost.

In reinforcement learning, the choice of which batch of tasks to perform each time is considered as an action, and if the average time-out per task is smaller, the better the action is proved to be. However, our earlier actions affect the later decisions, because each decision directly affects the state of the threads in each virtual machine. We view the decision process as a tree, each path in the tree represents a different choice for us. The tree is shown in Figure 2.

The root node is the moment when no tasks have been processed yet, and each time we process a batch of tasks, it represents the selection of the next level of nodes in a tree. When we have reached the leaf node, it proves that all the tasks have been processed. All of our previous processes can be seen as a path starting from the root node in the tree all the way to the leaf nodes, and we are guided by reinforcement learning to choose one path at a time, and the path we choose is our one action.

Each of our episode refinds a path from the root node in the tree to a leaf nodes, at which point the state of the virtual machines in any path taken is consistent; because we choose a path that has been taken before, it means that we choose the same task each time as before. This step can greatly reduce the range explored by our agent. This is necessary in scheduling problems where we sacrifice a small amount of space for exploration and can be extremely efficient. In

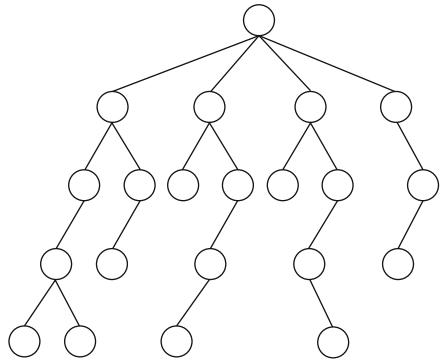

Figure 2: The decision-making process for task selection.

the next subsection, we will show the outstanding performance of EATS in the dataset.

## 4. Experimental

In this section, we conduct experiments to evaluate EATS in IoT system of various performance metrics and analyze the results.

*4.1. Experiment Setup.* All the algorithms are run on Ubuntu 16.04.5 Linux Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40 GHz, 256 GB RAM. To verify the effectiveness of EATS, experiments were set up to test it in different environments. In the system, we have 6000 tasks that arrive in 1000 seconds. We vary the difficulty of the scheduling problem by dynamically adjusting the number of threads per virtual machine and the efficiency of each virtual machine, with more threads and higher efficiency representing a less difficult scheduling problem.

To test the effectiveness of the proposed algorithm EATS in IoT systems, we constructed different scenarios and chose two baseline algorithms for this article:

(1) The MIN-MIN algorithm [13], which is a classical algorithm used in scheduling problems. This algorithm focuses on the best solution of the current task in each moment, and when the new task arrives, it assigns it to the thread in the virtual machine that can complete it first. There is also an algorithm similar to it, MAX-MIN [14], which prioritises the tasks that take the most time to complete. In our dataset, the MIN-MIN algorithm completely dominates the MAX-MIN algorithm, so we will not show it here

(2) The random algorithm (RDA); although RDA does not have amazing ideas, it can still give good solutions in some cases. In most cases, RDA is not competitive with state-of-the-art algorithms, but it is a good baseline that accurately reflects the difficulty of the dataset. In our work, the amount of time-out generated by the random strategy and the proportion of time-out tasks are obtained by averaging the results of three times

*4.2. Experiment Results.* For the problem of the task scheduling in IoT systems, we evaluate the results of the different algorithm from two perspectives: (1) time-out (the lower the time-out, the better the performance of the algorithm) and (2) the proportion of time-out tasks (the lower the value, the better the performance of the algorithm, where the proportion of time-out tasks is the number of tasks that are currently time-out divided by the number of tasks that have been processed so far).

We compare the experimental results of our proposed EATS with those of the baseline algorithm in different scenarios. With Figures 3 and 4, we can see how the amount of time-out tasks and the proportion of time-out tasks change as the task number increases for EATS and its competitors. In our experiments, the amount of time-outs is in seconds. Observed from the results of Figure 3, the performance of our proposed algorithm and MIN-MIN totally dominates RDA. Although RDA is not competitive with the other algorithms, it gives a clear picture of the difficulty of the set of tasks. By looking at the above graph, we see that both the EATS and MIN-MIN algorithms have outstanding performance when the number of tasks is small. However, the performance of the MIN-MIN algorithm degrades as the number of tasks in IoT systems continues to increase. When 500 tasks were generated, EATS started to experience a time-out amount.

As can be seen in Figures 3 and 4, the performance gap between the different algorithms becomes more pronounced when the number of tasks increases. The poor performance of RDA in this set of environments also indicates that the set of tasks which need be scheduled is difficult to handle. The harder cases tend to be closer to the real world, while also showing performance differences between algorithms.

When processing the full task, we can see through Figure 3 that the MIN-MIN algorithm has almost twice as many time-outs as our algorithm, but in Figure 4, we can see that the percentage of task time-outs for the MIN-MIN algorithm does not reach twice that of our algorithm, indicating that our algorithm is forward-looking enough that it may choose to let the current task time-out but ensures that the total time-out is as small as possible. This foresight is important in large-scale task scheduling problems. When tasks become larger, it is a poor performance to focus more on the task at hand.

In order to show the effect of proposed EATS in different environments, we test 2000 tasks generated in IoT systems under the condition that the virtual machine efficiency is 1/2 of normal. We adjusted the number of different threads in the virtual machine. As shown in Figures 5 and 6, when the number of threads in virtual machine increases, this problem becomes simpler. As the RDA algorithm has a much higher amount of time-outs in this scenario, we do not show its time-outs in Figure 5. Even though this set of scenarios only handles 2000 tasks, it is extremely difficult. We can see that the lower the number of threads in the virtual machine, the more difficult the problem becomes. When the problem becomes difficult, the efficiency gap between us and other algorithms also becomes larger, which shows that our proposed algorithm has outstanding performance in different scenarios.

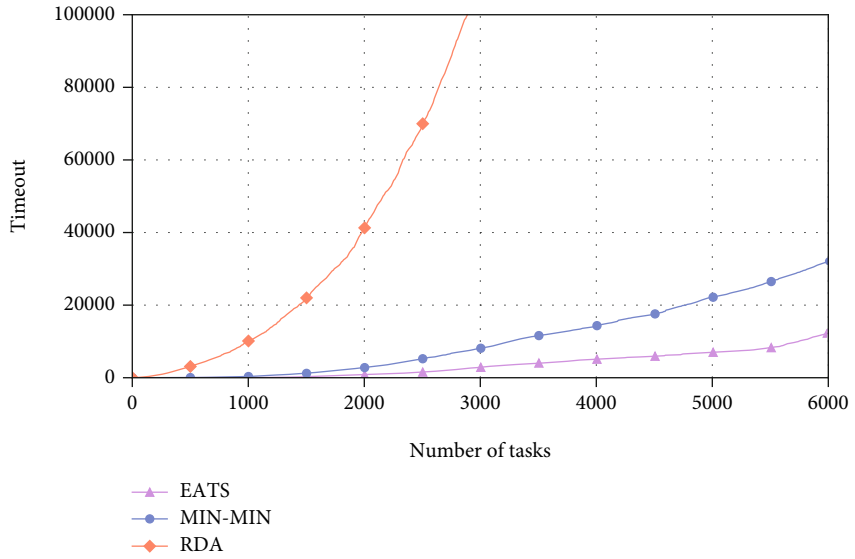


Figure 3: The amount of time-out changes as the number of tasks increases.

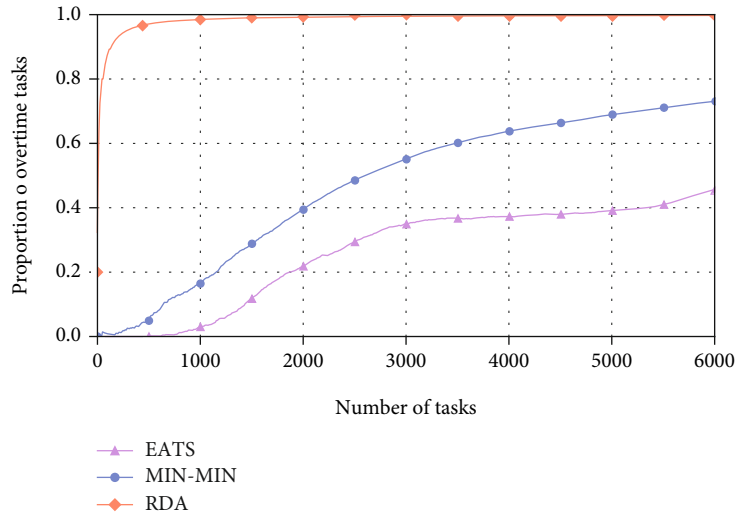


Figure 4: The proportion of overtime tasks changes as the number of tasks increases.

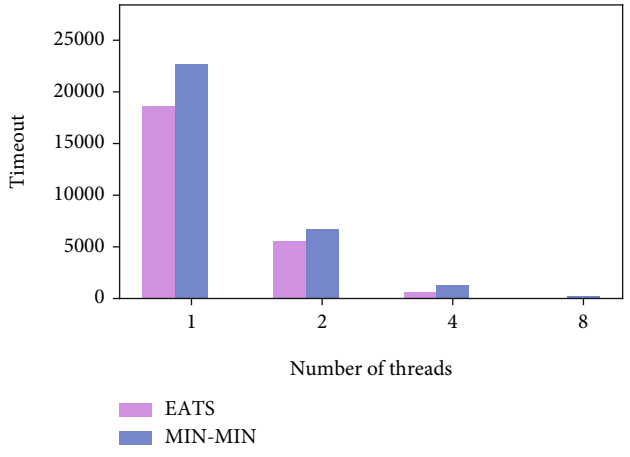


Figure 5: The amount of time-out changes with different threads on the virtual machine.

Not only do changes in the number of threads affect the difficulty of the scheduling problem but also changes in the efficiency of the virtual machine can make the problem difficult. In order to better demonstrate the practicability of our proposed algorithm, we tested the relationship between the virtual machine efficiency and the amount and proportion of task time-out, as shown in Figures 7 and 8.

As we can see, the problem becomes more difficult when the virtual machine becomes inefficient. When the virtual machine efficiency becomes 1/5 of the original, the proportion of time-out tasks has exceeded half. But that does not seem to be as big an impact as the reduction in threads. This is because when there are fewer threads, scheduling is not only more difficult but also affects the number of tasks that can be processed simultaneously. However, when the efficiency of the virtual machine becomes low, it will not affect the effective decision of the algorithm directly.

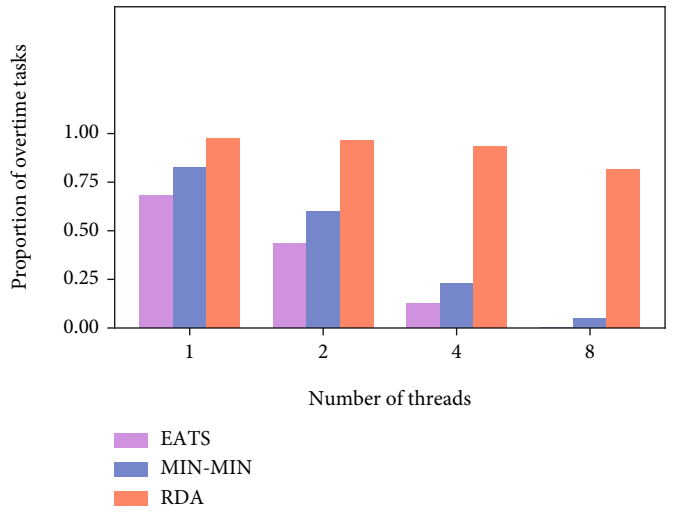


Figure 6: The proportion of time-out tasks changes with different threads on the virtual machine.

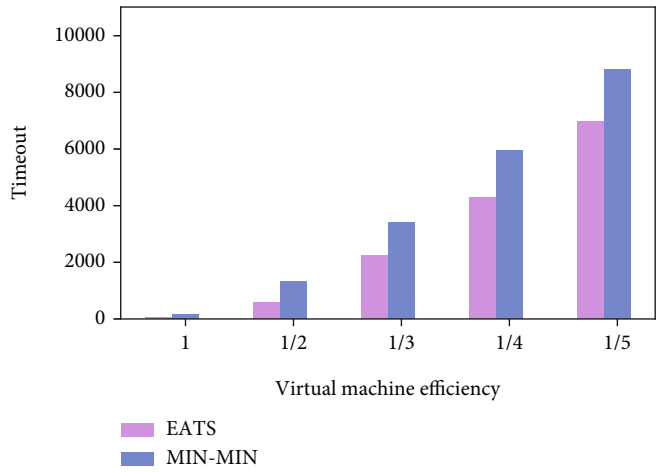


Figure 7: Relationship between the proportion of time-out tasks and virtual machine efficiency.

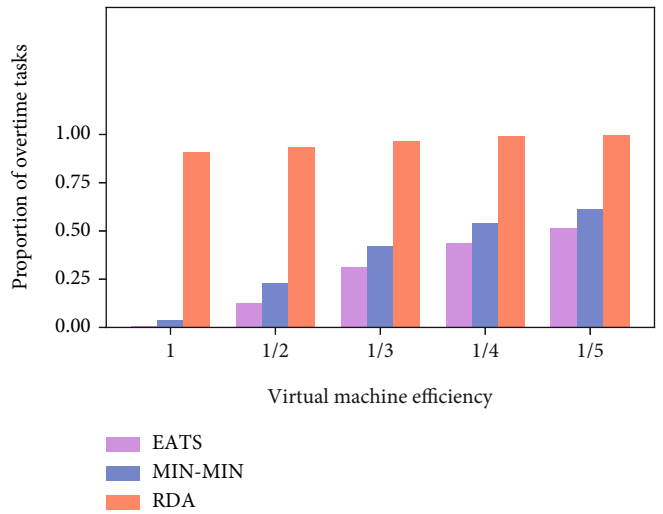


Figure 8: Relationship between time-out and virtual machine efficiency.

The experimental findings show that, in various environments, our proposed algorithm EATS performs better than the comparison algorithm. Additionally, the relationship between time-out amount and the proportion of time-out tasks demonstrates that EATS is adequately forward-looking. It does a good job of dropping the optimal case of the current task but guarantees the least total time-out for all tasks.

Different from other algorithms, we focus on the scheduling scheme of a batch of tasks at a time, and it will select some tasks to time out among a batch of tasks, but the overall impact is positive. Second, reinforcement learning guides the process to make this advantage even more obvious.

Although the RDA algorithm is not competitive with the state-of-the-art algorithms, we demonstrate the difficulty of the dataset through the performance of RDA. Our proposed EATS also has outstanding results in difficult scenarios, which shows that our proposed algorithm is efficient and practical.

## 5. Related Work

With the rapid development of IoT systems, IoT systems are widely used in different scenarios. Recently, Chen et al. [15] proposed a game theory approach in QOS aware computing offloading of IoT in LEO satellite edge computing, which can better deal with complex scenarios. Moreover, Boursianis et al. [16] assisted precision agriculture intelligent irrigation system through the Internet of Things platform. This article also focuses on issues in IoT systems, which are often used in cloud computing environments. With the development of the Internet of Things technology, many problems in the Internet of Vehicles can be better solved [17, 18].

There are many areas that deal with scheduling and resource allocation. In the virtual machine resource allocation problem, Li et al. [19] used reinforcement learning method to approximate the optimal allocation strategy based on the feedback state and reward. Li et al. [20] decomposed the transformed problem into subproblems in the resource allocation problem of IoT devices in smart buildings and solved them by stochastic optimization techniques. In addition to this, many improved methods for task scheduling have been proposed in recent years [21].

Cloud computing is an important platform to support IoT applications. Cloud computing is an important platform for supporting IoT applications, where edge computing is no less important than task scheduling in the cloud. In our previous work, there were a lot of research on edge computing. Chen et al. [22] studied in the edge caching of IoT services. They proposed non-orthogonal multiple access (NOMA) technology to improve the efficiency of resource transmission and reformulated the optimization problem as non-cooperative game model. Wu et al. [23] are driven by edge computing for target detection and image enhancement. Chen et al. [24] have focused on the unloading problem in edge cloud systems and proposed the idea of game-based decentralized task offloading (GDTO) to obtain offloading strategy and analyze the upper bound for the convergence time.

In cloud computing and IoT problems, many of them are difficult and it is crucial that we need to dynamically adapt the decisions we make. An approach for data security in the IoT is proposed by Cai et al. and Cai and Zheng [25, 26]. In this article, we also dynamically select the task to be computed each time by means of a reinforcement learning.

Recently, Wan et al. [27] have proposed edge computing-based preprocessing methods that can effectively reduce the demand on the cloud. You et al. [28] focus on joint task scheduling problem in mobile edge computing, which divides the problem into multiple subproblems, and the authors define an optimization problem in order to minimize the overall energy consumption of all UAVs. The continuous convex approximation technique and the branch-and-bound method are used to solve the high-quality solutions of the subproblem. Similarly, EATS also focus on the optimal solution of the subproblem, select the subproblem for each computation by means of a reinforcement learning algorithm, and process the subproblem by means of a combined optimization algorithm. We devise an efficient algorithm for knowing the arrival of a task in advance.

Finally, we need to model the problem, which we do by modelling the tasks and idle virtual machines in the IoT system as a bipartite graph. There are many ways of modelling different problems [29]. Chen et al. [30] proposed a flood prediction model of BiGRU with attention mechanism based on IoT system. In the industrial Internet of Things (IIoT), Huang et al. [31] built a Markov queuing model that captures the dynamics of IoT devices and edge servers and designed intelligent computing methods. A differential-private framework to predicting traffic flow was proposed by Cai et al. [32]. In the future, we will consider additional modelling approaches to better deal with problems in IoT systems.

## 6. Conclusions

In this article, we focus on the problem of task scheduling in large-scale IoT systems. We propose a novel reinforcement learning algorithm EATS that incorporates a combinatorial optimization algorithm and prove its lower bound. The experimental results show that EATS proposed in this paper has outstanding performance in different environments. In our future work, we will further focus on the performance of reinforcement learning algorithms in other IoT applications.

## 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 conflict of interest regarding the publication of this paper.

## Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (61902029), Project of Cultivation for Young Top-Notch Talents of Beijing Municipal Institutions (BPHR202203225), R&D Program of Beijing Municipal Education Commission (No. KM202011232015), Project for Acceleration of University Classification Development (Nos. 5112211036, 5112211037, and 5112211038), and BISTU College Students Innovation and Entrepreneurship Training Program (No. 5112210832).

## References

[1] J. Zhou, Y. Wang, K. Ota, and M. Dong, "AAIoT: Accelerating artificial intelligence in IoT systems," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 825–828, 2019.

[2] I. T. Christou, N. Kefalakis, A. Zalonis, and J. Soldatos, "Predictive and explainable machine learning for industrial internet of things applications," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 213–218, Marina del Rey, CA, USA, 2020.

[3] 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.

[4] X. Liu and X. Zhang, "Rate and energy efficiency improvements for 5g-based IoT with simultaneous transfer," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 5971–5980, 2018.

[5] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1122–1135, 2020.

[6] Y. Chen, W. Gu, and J. Xu, "Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning," *China Communications*, 2022.

[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, 2017.

[8] M. M. Drugan, "Reinforcement learning versus evolutionary computation: a survey on hybrid algorithms," *Swarm and Evolutionary Computation*, vol. 44, pp. 228–246, 2019.

[9] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the tenth international conference on Information and knowledge management*, pp. 25–32, Atlanta, GA, USA, 2001.

[10] C. Jin, J. Xu, Y. Han, J. Hu, Y. Chen, and J. Huang, "Efficient delay-aware task scheduling for IoT devices in mobile cloud computing," *Mobile Information Systems*, vol. 2022, Article ID 1849877, 10 pages, 2022.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[12] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[13] 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.

[14] U. Bhoi and P. N. Ramanuj, "Enhanced max-min task scheduling algorithm in cloud computing," *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, vol. 2, no. 4, pp. 259–264, 2013.

[15] Y. Chen, J. Hu, J. Zhao, and G. Min, "Qos-aware computation offloading in LEO satellite edge computing for IoT: a game-theoretical approach," *Chinese Journal of Electronics*, 2023.

[16] A. D. Boursianis, M. S. Papadopoulou, A. Gotsis et al., "Smart irrigation system for precision agriculture—the arethou5a IoT platform," *IEEE Sensors Journal*, vol. 21, no. 16, pp. 17539–17547, 2020.

[17] S. Zhang, J. Li, H. Luo, J. Gao, L. Zhao, and X. S. Shen, "Low-latency and fresh content provision in information-centric vehicular networks," *IEEE Transactions on Mobile Computing*, vol. 21, 2020.

[18] C. Chen, H. Li, H. Li, R. Fu, Y. Liu, and S. Wan, "Efficiency and fairness oriented dynamic task offloading in internet of vehicles," *IEEE Transactions on Green Communications and Networking*, vol. 6, 2022.

[19] 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.

[20] K. Li, J. Zhao, J. Hu, and Y. Chen, "Dynamic energy efficient task offloading and resource allocation for NOMA- enabled IoT in smart buildings and environment," *Building and Environment*, vol. 226, article 109513, 2022.

[21] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: a literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.

[22] Y. Chen, H. Xing, Z. Ma, X. Chen, and J. Huang, "Cost-efficient edge caching for noma-enabled iot services," *China Communications*, 2022.

[23] Y. Wu, H. Guo, C. Chakraborty, M. Khosravi, S. Berretti, and S. Wan, "Edge computing driven low-light image dynamic enhancement for object detection," *IEEE Transactions on Network Science and Engineering*, 2022.

[24] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. S. Shen, "Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: a game-theoretical approach," *IEEE Transactions on Mobile Computing*, pp. 1–17, 2022.

[25] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 577–590, 2016.

[26] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart cyber-physical systems," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 2, pp. 766–775, 2018.

[27] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles," *Pattern Recognition*, vol. 121, article 108146, 2022.

[28] 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.

[29] L. Zhao, C. Wang, K. Zhao, D. Tarchi, S. Wan, and N. Kumar, "Interlink: a digital twin-assisted storage strategy for satellite-terrestrial networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, 2022.

[30] C. Chen, J. Jiang, Y. Zhou, N. Lv, X. Liang, and S. Wan, "An edge intelligence empowered flooding process prediction using internet of things in smart city," *Journal of Parallel and Distributed Computing*, vol. 165, pp. 66–78, 2022.

[31] J. Huang, H. Gao, S. Wan, and Y. Chen, "AoI-aware energy control and computation offloading for industrial IoT," *Future Generation Computer Systems*, vol. 139, 2022.

[32] Z. Cai, X. Zheng, and J. Yu, "A differential-private framework for urban traffic flows estimation via taxi companies," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, pp. 6492–6499, 2019.