mathematics

Article

Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing

Yugui Wang [1,2], Shizhong Dong [3,*] and Weibei Fan [1]

1 School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China; 2022010302@njupt.edu.cn or 20140002@njpi.edu.cn (Y.W.); wbfan@njupt.edu.cn (W.F.)
2 School of Information Engineering, Nanjing Polytechnic Institute, Nanjing 210048, China
3 Wuhan Academy of Social Sciences, Wuhan 430019, China
* Correspondence: dongshizhong5@gmail.com

Abstract: The explosive growth of users and applications in IoT environments has promoted the development of cloud computing. In the cloud computing environment, task scheduling plays a crucial role in optimizing resource utilization and improving overall performance. However, effective task scheduling remains a key challenge. Traditional task scheduling algorithms often rely on static heuristics or manual configuration, limiting their adaptability and efficiency. To overcome these limitations, there is increasing interest in applying reinforcement learning techniques for dynamic and intelligent task scheduling in cloud computing. How can reinforcement learning be applied to task scheduling in cloud computing? What are the benefits of using reinforcement learning-based methods compared to traditional scheduling mechanisms? How does reinforcement learning optimize resource allocation and improve overall efficiency? Addressing these questions, in this paper, we propose a Q-learning-based Multi-Task Scheduling Framework (QMTSF). This framework consists of two stages: First, tasks are dynamically allocated to suitable servers in the cloud environment based on the type of servers. Second, an improved Q-learning algorithm called UCB-based Q-Reinforcement Learning (UQRL) is used on each server to assign tasks to a Virtual Machine (VM). The agent makes intelligent decisions based on past experiences and interactions with the environment. In addition, the agent learns from rewards and punishments to formulate the optimal task allocation strategy and schedule tasks on different VMs. The goal is to minimize the total makespan and average processing time of tasks while ensuring task deadlines. We conducted simulation experiments to evaluate the performance of the proposed mechanism compared to traditional scheduling methods such as Particle Swarm Optimization (PSO), random, and Round-Robin (RR). The experimental results demonstrate that the proposed QMTSF scheduling framework outperforms other scheduling mechanisms in terms of the makespan and average task processing time.

Keywords: reinforcement learning; Q-learning; upper confidence bound; task scheduling

MSC: 68U01

check for updates

Citation: Wang, Y.; Dong, S.; Fan, W. Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing. *Mathematics* **2023**, *11*, 3364. https://doi.org/10.3390/math11153364

Academic Editors: Jou-Ming Chang, Ling-Ju Hung and Chia-Wei Lee

Received: 26 June 2023
Revised: 26 July 2023
Accepted: 30 July 2023
Published: 1 August 2023

1. Introduction

With the rapid development of the Internet of Things (IoT) we have entered a new era of digitalization connecting various smart devices and sensors and enabling interconnectivity and information sharing among them. However, the rapid growth of the IoT brings with it a series of challenges and issues, one of which is effectively handling the massive influx of task requests generated by devices. In the IoT, a vast number of devices and sensors generate an enormous amount of data processing tasks that need to be collected, stored, analyzed, and processed. Traditional methods of task processing often struggle to cope with such a tremendous task scale and high task generation rate. Cloud computing technology provides robust support for addressing this problem. Based on a network, cloud computing offers efficient computing and storage resources, making the processing of large-scale tasks feasible. By connecting IoT devices to cloud platforms, tasks generated by the devices can be uploaded to the cloud in real-time for processing. Cloud infrastructure provides the necessary computational power, storage capacity, and scalability to efficiently handle the influx of tasks. This enables real-time and scalable processing of tasks generated by IoT devices, facilitating data analysis and decision-making while providing valuable insights. Cloud computing empowers the IoT ecosystem by offloading the computational burden from individual devices to the cloud, ensuring efficient task management and enabling advanced data processing and analytics. The integration of IoT and cloud computing facilitates the seamless and effective handling of the vast volume of tasks generated by IoT devices, paving the way for enhanced IoT applications and services in the digital age.

A vast amount of resources such as computing power, storage capacity, and network bandwidth are available in the cloud computing environment. However, these resources need to be allocated reasonably among multiple users and tasks in order to meet the needs and priorities of different users. Resource allocation in the dynamic and heterogeneous cloud computing environment can become highly complex and challenging. First, tasks and user demands in the cloud computing environment exhibit great diversity and variability; different tasks may require different types and scales of resources, and user demands for resources can change over time. Second, resources in the cloud computing environment are limited. Due to the scarcity of resources, it is necessary to allocate resources effectively in order to maximize user satisfaction and provide high-performance services. Load balancing of resources needs to be considered to avoid performance degradation caused by resource overload; moreover, resource conflicts and competition inevitably exist in the cloud computing environment. Multiple tasks or users may compete for the same resources simultaneously, leading to resource contention and increased delays. Therefore, an effective resource allocation scheduling strategy is needed in order to address resource conflicts and provide fair and efficient resource utilization.

To address the performance issues in task scheduling [1–3], researchers have proposed various novel research methods. Existing cloud computing task scheduling methods typically fall into the following categories. (1) Static Scheduling Methods: these methods determine the execution order and allocation scheme of tasks before task submission, such as Shortest Job First (SJF), Earliest Deadline First (EDF), and Minimum Remaining Time (MRT). While these methods are simple to implement, they cannot adapt to dynamic environments and changing task demands. (2) Heuristic Scheduling Methods: these methods make task scheduling decisions based on past experience and on rules such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). These methods consider certain task priorities and resource utilization efficiency; however, they lack global optimization and dynamic adjustment capabilities. (3) Load Balancing Scheduling Methods: the aim of these methods is to achieve load balancing among devices in order to improve resource utilization and system performance. Examples include round-robin (RR) scheduling, random scheduling, and queue-length-based scheduling. These methods can balance task loads, but fail to consider task characteristics and device performance differences.

Existing cloud computing task scheduling faces several challenges. (1) Large-Scale Task Processing: cloud computing environments may have a massive number of tasks that need to be scheduled and processed. Handling such a large-scale task processing introduces complexity in task scheduling and increases computational complexity. (2) Dynamic Environment: task arrivals and departures in the cloud computing environment are dynamically changing. The number of tasks and their resource requirements can change at any time. Therefore, task scheduling needs to exhibit real-time adaptability and flexibility to promptly respond to changing demands. (3) Resource Allocation and Load Balancing: task scheduling requires the rational allocation and utilization of resources in the cloud computing environment in order to achieve load balancing and maximize resource utilization. Inadequate resource allocation or load imbalance can lead to system performance degradation and resource wastage.

Based on the need for better task scheduling performance, this study applies reinforcement learning to cloud computing task scheduling and proposes a two-stage dynamic cloud task scheduling framework called Q-Learning-based Multi-Task Scheduling Framework (QMTSF). The first stage occurs in the cloud data center,; when user tasks arrive, they and are initially stored in a global queue (GQ). Then, based on the characteristics of the tasks, they are dynamically assigned to appropriate servers. The second stage takes place within each server, specifically, the server queue (SQ). A time window is used to determine when the tasks in the queue are assigned to each virtual machine (VM). For each time window, the server's task scheduler first prioritizes all tasks in the queue based on their deadline constraints, then employs a UCB-based Q-Reinforcement Learning (UQRL) strategy to assign tasks to suitable VMs. An incentive mechanism is applied to reward assignments that minimize the makespan of tasks.

The main contributions of this paper are as follows:

(1) We propose Q-learning-based Multi-Task Scheduling Framework (QMTSF) to address task scheduling issues in cloud computing. Task assignment is performed in the cloud data center, while task scheduling for different VMs is conducted within each server.
(2) For the task assignment phase in the cloud data center, we introduce a method based on server task type priorities to allocate tasks to suitable servers.
(3) For task scheduling within each server, we first employ a dynamic sorting strategy to prioritize tasks based on their deadlines. This enables the Q-Learning-based task scheduler to consider the more urgent tasks earlier, contributing to improved service quality. Additionally, we apply an improved Q-Learning algorithm based on the UCB strategy (UQRL) to the task scheduling within each server, resulting in reduced average task processing times and overall makespan.
(4) We conducted experiments to validate the effectiveness and superiority of QMTSF, comparing its performance against the PSO algorithm, random scheduling, and RR scheduling. The experimental results confirm the effectiveness of QMTSF.

The structure of this paper is as follows: Section 2 provides an overview of the current research progress and achievements in cloud computing task scheduling; Section 3 introduces the QMTSF system model proposed in this paper; Section 4 presents the dynamic task allocation algorithm and UQRL algorithm used in QMTSF; and Section 5 presents the results of the simulation experiments. Finally, Section 6 concludes the paper.

2. Related Works

Task scheduling is crucial for the efficient operation of cloud computing. A good task scheduling strategy can effectively reduce the execution time of user tasks and meet their demands under different constraints, increase the utilization of cloud resources, and reduce energy consumption and operational costs.

Traditional scheduling strategies such as FCFS [4], RR [5], and random [6] can be used as task scheduling solutions; however, their performance is no longer sufficient to meet the requirements of cloud computing. Heuristic algorithms are designed based on the evolutionary principles of biological organisms through simulations of genetic operations, mutation, and natural selection to find optimal solutions, and have been applied to task scheduling in cloud computing. Heuristic algorithms have shown significant effectiveness in reducing task execution time, improving resource utilization, reducing energy consumption, and increasing throughput in cloud computing task scheduling. In a cloud data center, inefficient task scheduling can lead to insufficient resource utilization and decreased revenue. In such cases, reducing the makespan becomes crucial for performing efficient task scheduling in the cloud. In a study by Raju et al. [7], a hybrid algorithm [8] combining Ant Colony Optimization (ACO) and CSA was proposed as a resource scheduling strategy to reduce the makespan. This hybrid algorithm helped to minimize completion time or makespan by optimally allocating the required resources for jobs submitted on time. To address the optimization problem in workload scheduling in cloud computing, Khalili et al. [9] proposed a single-objective Particle Swarm Optimization (PSO) [10] algorithm. Different inertia weight strategies were combined in the PSO to minimize the makespan. The results indicate that combining the Linearly Decreasing Inertia Weight (LDIW) with the PSO can reduce the makespan. In the same context, Gabi et al. [11] introduced a traditional CSO task scheduling technique incorporating the LDIW equation to overcome the trapping problem caused by local search. Using the CloudSim simulator tool, this technique efficiently maps tasks to VMs based on an enhanced convergence speed, thereby minimizing the completion time. For the current architecture of cloud data centers, Sharma et al. [12] proposed a new Hybrid Coordinated-heuristic Genetic Algorithm (HIGA) [13] solution for real-world scientific workflows to address energy-efficient task scheduling. By combining the search capability of Genetic Algorithms (GA) and the development capability of Harmony Search (HS), HIGA provides intelligent awareness of the local and global search space, leading to fast convergence. The goal of this work was to minimize completion time, computational energy, energy consumption of resources, and overhead associated with the scheduler. Wu et al. [14] proposed a market-oriented resource scheduling technique based on ACO, PSO, and GA considering dynamic resource scheduling at both the task level and the service level. This approach optimizes the makespan and CPU time, reducing the overall operating costs of the data center. Meshkati et al. [15] proposed a Hybrid ABC-PSO Scheduling Framework (HSF) that, in addition to managing the position of VMs on physical nodes, reduces active nodes by turning off unused ones, thereby saving energy. Song et al. [16] designed a Fast N-Particle Swarm Optimization (FN-PSO) algorithm to address the scheduling problem between composite tasks in a distributed cloud computing environment, reducing task execution time. Huang et al. [17] proposed a PSO-based task scheduler that utilizes a logarithmic decrement strategy to minimize the makespan of the cloud system in cloud task scheduling. Zhou et al. [18] proposed an improved GA combined with a greedy strategy to optimize the task scheduling process, reducing the number of algorithm iterations and significantly improving performance in terms of minimizing the total completion time, average response time, and QoS. Sulaiman et al. [19] introduced a hybrid heuristic algorithm based on genetic algorithms and lists for scheduling heterogeneous computing tasks. However, these optimization methods provide an optimal solution and are primarily designed for static optimization problems, making them less suitable for dynamic optimization problems.

In response to the aforementioned issues, Reinforcement Learning (RL) [20–22] has been actively applied in cloud computing task scheduling. RL is a direct learning strategy that enables an agent to learn appropriate policies through trial and error, acquiring rewards from the environment without any prior knowledge. The agent observes its own state and selects actions to obtain rewards associated with the chosen actions from the environment. Based on past experiences, the agent learns the optimal actions to maximize rewards. Q-Learning is a classic RL algorithm that has been applied in task scheduling. Wei et al. [23] and Khan et al. [24] proposed a sensor node cooperative RL approach for task scheduling to optimize the trade-off between energy and application performance in sensor networks. Each node considers the local state observations of its neighbors and shares knowledge with other agents to enhance the learning process and achieve better results than individual agents. Wei et al. [25] combined the Q-Learning RL method with an improved supervised learning model, Support Vector Machine (ISVM-Q), for task scheduling in Wireless Sensor Networks (WSNs). The ISVM model takes state–action pairs as inputs and calculates Q-value estimates. Based on this estimation, the agent selects the optimal tasks to be performed. Experimental results have demonstrated that the ISVM method improves application performance by putting sensors and communication modules in sleep mode when necessary while preserving network energy. Li et al. [26] proposed Q-Grid, an RL-based hierarchical protocol, to improve information dissemination rates by minimizing latency and hop count. The protocol divides the geographic region into small grids and searches for the optimal grid next to the destination based on the traffic of neighboring grids, establishing a Q-Value table. Ding et al. [27] proposed QEEC, an efficient energy-saving cloud computing task scheduling framework based on Q-Learning. This framework utilizes the M/M/S queue model for task reallocation and employs a Q-Learning algorithm based on an Ît-Greedy policy [28] to achieve energy-efficient utilization of cloud resources during task scheduling. Ge et al. [29] introduced a global-perspective flexible task scheduling scheme based on Q-Learning, aiming to improve task scheduling success rate, reduce latency, and prolong the lifespan of the Internet of Things (IoT). Liu et al. [30] proposed a Q-Learning approach based on multi-objective optimization and the FANET routing protocol. They were able to achieve a higher packet delivery rate and lower latency and energy consumption, leading to better performance.

For RL, the process can be divided into exploration and exploitation. Exploration refers to the selection of actions to gain new knowledge, while exploitation refers to selecting the best action based on stored knowledge. The agent should strike a balance between these two. Resolving this trade-off problem allows the agent to achieve more efficient learning, as an inefficient exploration strategy would result in more suboptimal actions and poorer performance. In the exploration phase of Q-Learning, the ε-Greedy algorithm is commonly used, which employs a completely random policy for exploration. However, this algorithm converges slowly and is prone to becoming trapped in local optima. In the exploration–exploitation trade-off, only the immediate benefits are considered, without taking into account the cost of obtaining those benefits (i.e., exploration attempts). Excessive consideration of immediate benefits may overlook the value of other potential strategies, while excessive focus on exploration attempts may neglect the maximum benefit from exploitation, leading to a lower overall success rate. Thus, a good choice is to balance current benefits and exploration attempts with the aim of achieve the highest possible success rate within a limited number of rounds.

The Upper Confidence Bound (UCB) algorithm is an algorithm that uses the current benefits and exploration attempts as the criteria for selection. The UCB strategy was proposed as a method of balancing exploration and exploitation in the multi-armed bandit problem [31,32]. This method balances the relationship between exploration and exploitation by considering the number of times an action is chosen, and has better performance [33] than methods such as ε-Greedy. Several improved methods based on UCB have been proposed, such as UCB1-tuned [34] and Discounted UCB [35]. The UCB approach adopts an optimistic attitude towards uncertainty, and uses the upper confidence bound of option returns as the estimated value of returns. The basic idea is that the more times a certain option is tried, the narrower the confidence interval for the estimated returns becomes, and the lower the uncertainty. Options with higher mean values tend to be selected multiple times, representing the conservative part of the algorithm (exploitation). On the other hand, options with fewer attempts have wider confidence intervals and higher uncertainty. Options with wider confidence intervals are more likely to be chosen multiple times, representing the aggressive part of the algorithm (exploration).

Saito et al. [36] studied the performance of applying UCB to the Q-Learning algorithm and conducted numerical analysis, demonstrating its balance between exploitation and exploration and its superior performance compared to other methods. Yu et al. [37] applied UCB-2 to the Q-Learning algorithm, allowing multiple agents to act in parallel and achieving a near-linear speedup in execution time. Their approach outperformed existing parallel algorithms in terms of sample complexity. Bae et al. [38] formulated network optimization as a Markov decision process problem and developed a UCB-based exploration reinforcement learning algorithm called Q^{+}-Learning. This approach improves convergence speed, minimizes performance loss during the learning process, and achieves optimal throughput. Elsayed et al. [39] utilized an energy-efficient Q-Learning algorithm based on the UCB strategy to dynamically activate and deactivate SCCs (sub-carrier components) in order to maximize user throughput with the minimum number of active SCCs, thereby maximizing energy efficiency. Li et al. [40] proposed a user scheduling algorithm based on the UCB strategy in reinforcement learning that continuously updates the estimated behavioral value of each user. This approach avoids the phenomenon of “extreme unfairness” during the exploration phase, reduces algorithm complexity, and improves system throughput. Yu et al. [41] introduced a variant of Q-Learning called Q-greedyUCB, which combines the average reward algorithm of Q-Learning and the UCB algorithm to achieve an optimal delay–power trade-off scheduling strategy in communication systems. Simulation results demonstrate that this algorithm is more effective than ε-greedy and standard Q-Learning in terms of cumulative reward and convergence speed.

3. System Model

Cloud computing, as a common way of resource allocation, has many optimization problems. This paper proposes a task scheduling framework called QMTSF based on reinforcement learning to solve the task allocation and scheduling problems in cloud computing environments. We define the cloud environment scenario as follows. There are n Servers in a cloud center, represented as $S_1, S_2, \ldots, S_i, \ldots, S_n$, where $1 \leq i \leq n$. For any server, the number of VMs is m, represented as $VM_1, VM_2, \ldots, VM_j, \ldots, VM_m$, where $1 \leq j \leq m$. QMTSF consists of two steps used to complete the scheduling of tasks to the VMs.

3.1. Stage 1: Task Allocation

In the first stage of task allocation, we first assign each task to a server that is suitable for processing that type of task.

As shown in Figure 1, A cloud data center can have a large number of servers, usually hundreds or thousands. Due to the nature of the cloud data center and the diversity of user requests, we chose to implement a scheduling program based on task type in the first stage of QMTSF. A GQ is used in the cloud data center to buffer all incoming user requests. The task dispatcher continuously monitors and manages unfinished user requests and takes timely action to assign them to each server node for processing. If the task assignment process fails, the task is re-added to GQ and waits for reassignment.

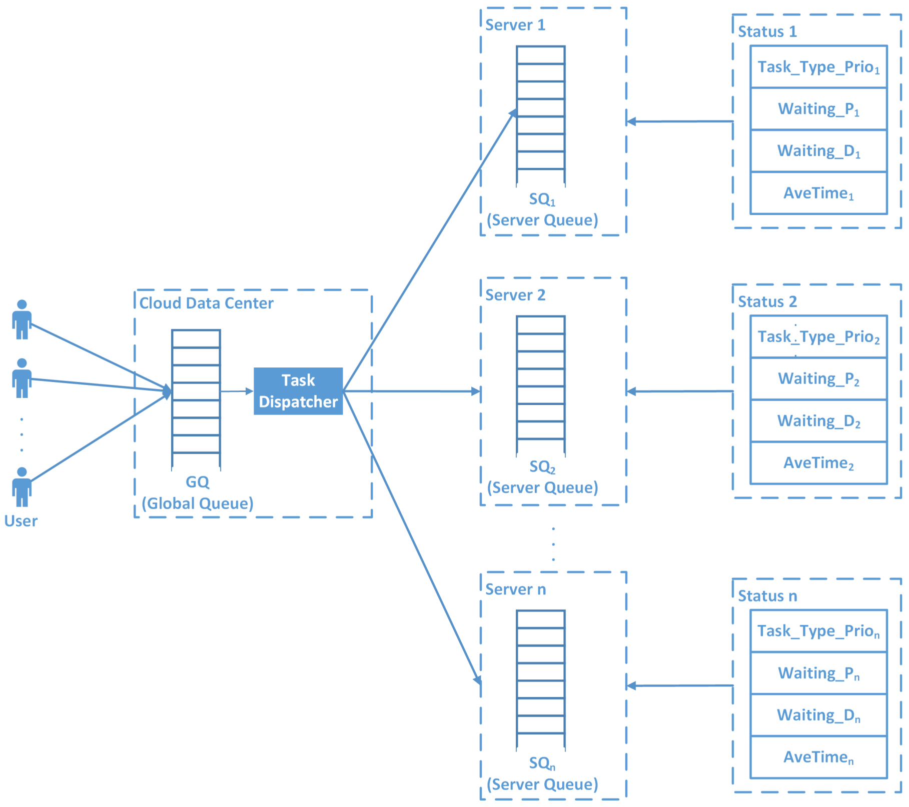

Figure 1. QMTSF stage one: task allocation.

In the cloud environment, multiple users generate different types of task processing requests. Each task that arrives at the cloud data center contains information, such as the task type, deadline, task size, etc. Due to the complexity and diversity of the cloud environment, there may be different types of user task requests, such as video tasks, voice tasks, text data analysis tasks, etc. How to allocate different task requests to suitable servers for processing is an important optimization problem in cloud computing.

As shown in Figure 1, in the first stage of QMTSF, tasks are classified and assigned initial values based on the actual situation of the cloud environment. Then, based on the specific information of each server in the cloud data center, such as CPU, memory, bandwidth, processing speed, network location, etc., the processing priority of each server for its task category is set. Finally, an update interface is provided in the system to update the task category processing priority of servers at any time when a new task type arrives. Task classification in the cloud environment is dynamically generated and updated. When a cloud data center receives a new task, it first looks up the *Task_Type_Prio* table to check whether the task type already exists. If it does, the task is allocated based on the algorithm described in Section 4. If the task type does not exist, the processing time for the task is calculated based on the task size and the processing speed of each server. Subsequently, priority values for this task type on different servers are generated, a new record is added to the *Task_Type_Prio* table for each server, and the value is updated for the entire table.

For example, in a cloud environment with L task categories, we set the task categories as follows: $VIDEO\_TASK = 0, VOICE\_TASK = 1, \ldots, TEXT\_TASK = L$. The task category priority of any server in the cloud data center is an L-dimensional tuple *Task_Type_Prio*, indicating which type of task the server is suitable for processing. In actual task allocation, the task category priority of each server is searched to find the most suitable server for processing the task and allocate the task to it. If all suitable servers are already allocated or if no suitable server is found, the task is allocated to the server with the next lower priority task category, and so on. Assuming that the priority tuple of a server i is $Task\_Type\_Prio_i = < L, 1, \ldots, 0 >$, this means that server i is most suitable for processing text data analysis tasks, followed by voice tasks, and least suitable for processing video tasks. Because the task classification of servers in the cloud environment is closely related to the actual situation of the cloud environment, in this article we focus on using reinforcement learning to schedule tasks when tasks are allocated from servers to VMs. Therefore, the actual situation of task classification is not described in detail in this article. However, we provide a priority update interface for the dynamic expansion of task categories in the later stage of implementation. A detailed description of the algorithm is provided in Section 4.

3.2. Stage 2: Task Scheduling

In the second stage of task scheduling, the tasks in the server cache queue are first dynamically sorted, and then the UQRL algorithm is used for task scheduling from the server to the VMs. The structure is shown in Figure 2.

Each server has m number of VMs, represented as $VM_1, VM_2, \ldots, VM_j, \ldots, VM_m$, where $1 \leq j \leq m$. Each VM has a task queue, called the virtual machine queue (VMQ), represented as $VMQ_1, VMQ_2, \ldots, VMQ_j, \ldots, VMQ_m$, where $1 \leq j \leq m$. When tasks arrive at the server, they are first cached in the SQ. We have defined a time window to execute the UQRL task scheduling algorithm once it is within a certain time period.

In existing Q-learning-based cloud task scheduling methods, user requests are usually processed using a first-come-first-served (FCFS) approach. However, actual submitted user tasks may have various requirements for task deadlines (completion times). Intuitively, all other conditions being equal, tasks with shorter deadlines should be scheduled first. Therefore, in this study, tasks in the SQ are dynamically sorted based on their deadline, making it easier to prioritize urgent tasks and improve the system's QoS. Then, the UQRL algorithm is used to schedule tasks and send them to the VMQs of the different VMs, where they wait to be executed. The VMs accept tasks one-by-one from their respective VMQs for execution. A detailed description of the algorithm is provided in Section 4.

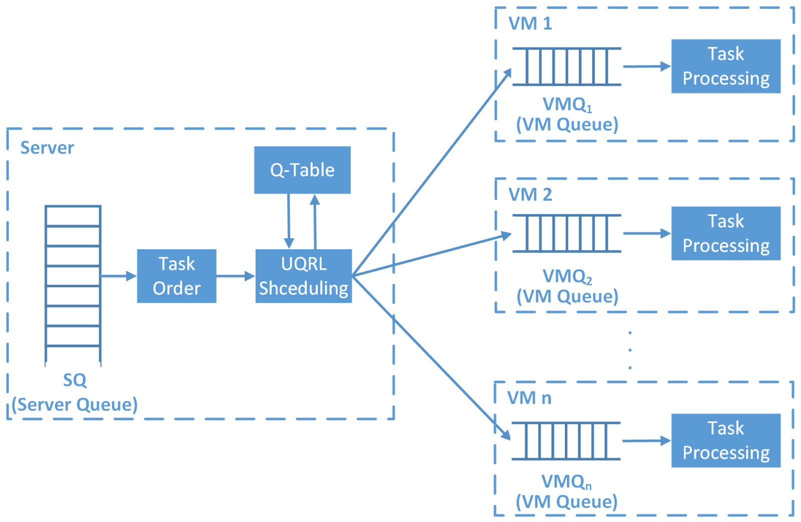

Figure 2. QMTSF stage two: task scheduling.

The time a task takes from entering the cloud center to completion can be divided into several parts: waiting time (the time spent in queue), transmission time (the time for a task to be transmitted over the link), and execution time (the time required for the VM to process the task). This article mainly discusses RL scheduling algorithms. To focus on the research problem, the transmission time is temporarily ignored, and only waiting time and execution time are considered. In the first stage of task allocation, when a user task arrives in the GQ it is immediately assigned to the SQ. When the number of servers is sufficient and the SQ capacity is large enough, the waiting time of tasks in the GQ can be ignored. Therefore, we only consider the waiting time of tasks in the SQ and VMQ.

For user tasks, we make the following assumptions:

(1) When user tasks arrive at the cloud server, they are independent.
(2) Each task is complete and cannot be further divided into smaller tasks.
(3) There are no dependencies between tasks.
(4) A VM can only execute one task at a time.

4. QMTSF Algorithm

4.1. Priority-Based Task Allocation Algorithm

Each server in the cloud computing environment maintains a server task queue (SQ), represented as $SQ_1, SQ_2, \ldots, SQ_i, \ldots, SQ_n$, which receives tasks assigned from the cloud data center. Each server has a status table Status, which includes information such as the task type priority tuple, the number of tasks waiting to be assigned in the server, the total number of tasks assigned to VMs but not yet executed, and the average processing time of tasks in the server. The cloud task scheduler assigns tasks to the appropriate SQ based on the task status in the GQ and the status table information Status of all servers, which then wait in the SQ for further task scheduling. For any server i, we use $Waiting\_D_i = |SQ_i|$ to indicate the number of tasks waiting in the queue and $Waiting\_P_i$ to indicate the total number of tasks assigned to VMs but not yet executed; $Task\_Type\_Prio_i$ is a task type priority tuple, and Ave_Time_i is the average processing time of completed tasks (including waiting time and processing time). Whenever a task is completed in server i, Ave_Time_i is dynamically updated. The workflow for task allocation to hosts in the cloud data center follows Algorithm 1:

Algorithm 1: Priority-based task allocation

```
1  Initialize: server number $n$, task number $|GQ|$, $ServerList = \phi$
2  While $|GQ|$ != 0 do
3      get first task $j$ and get attribute $Task\_Type_j$
4      for $i = 1, 2, \ldots, n$
5          if ($Task\_Type\_Prio_i[0] == Task\_Type_j$)
6              if $SQ_i$ is full
7                  // can't allocate task to this server
8                  continue;
9              Computing estimated completion time $Time\_E^i_j$
10             $ServerList.add < i, Time\_E^i_j >$;
11             continue;
12     if $ServerList$ != $\phi$
13         find the minimum $Time\_E^i_j$;
14         Assign $Task_j$ to $S_i$;
15         remove $Task_j$ from $GQ$;
```

In line 9 of the above algorithm, the expected completion time of each task j assigned to server i needs to be calculated; the formula for calculating this time is as follows:

$$Time\_E^i_j = (Waiting\_P_i + Waiting\_D_i + 1) * Ave\_Time_i$$

The estimated completion time for a task is related to the number of tasks waiting in the SQ and the number of tasks assigned but not yet processed. Here, Ave_Time_i is the average processing time of tasks completed on server i, including the waiting time and processing time. This time is updated automatically whenever a task is completed on the VM of server i. This design avoids the need to calculate the tasks in each VMQ, saving processing time on task assignment.

In line 12 of the algorithm, if ServerList is empty the system considers assigning the task to a suboptimal server, that is, $Task\_Type\_Prio_{i[1]} == Task\_Type_j$. In this case, lines 4 to 15 are executed again; if no suboptimal host is found, the task assignment fails and no lower priority server is considered. The task is reinserted at the end of the GQ and waits for the next assignment. Although task assignment failures do occasionally occur in practice, the probability is very low. The optimal and suboptimal cases can already cope with most environments; therefore, based on the actual situation and from the perspective of time saving, we do not consider assigning servers below the suboptimal level.

4.2. UQRL Algorithm

Reinforcement learning is an experiential machine learning technique where the agent acquires experience through a trial-and-error process to improve future choices. In this case, the problem to be solved is represented as a discrete-time stochastic control process known as a Markov Decision Process (MDP). Typically, a finite MDP is defined by a tuple (S, A, p, r), where S is the finite state space, A is the finite action space for each state $a \in A$, p is the state transition probability from state s to state $s^{'} \in S$ when taking action $a \in A$, and $r \in R$ represents the immediate reward obtained after performing action a. The main objective of the agent is to interact with its environment at each time step $t = 0, 1, 2, \ldots$ to find the optimal policy π^* that achieves the goal while maximizing the long-term cumulative reward. The policy π is a function that represents the strategy used by the RL agent. It takes a state s as input and returns the action a to be executed. The policy can be stochastic as well, in which case it returns a probability distribution over all possible actions instead of a single action a. A Q-Value table is constructed in the server to store the accumulated rewards obtained, and is iteratively updated using the Bellman equation, allowing the decision-maker to always choose the best action at each decision time, as shown in the following formula:

$$Q_{(s,a)} \leftarrow Q_{(s,a)} + \alpha \left[R + \gamma \max_{a' \in A(s')} Q_{(s',a')} - Q_{(s,a)} \right]$$

where α is the learning rate and $\gamma \in [0,1)$ the discount factor, which controls the importance of future rewards relative to the current reward. A larger γ places more emphasis on expected future returns. If $\gamma = 0$, the agent is "short-sighted" and only concerned with maximizing immediate rewards.

4.2.1. State Space

The state space S of the server is a finite set of possible states in which the VMs may exist. We take the number of waiting tasks in the current queue of all VMs on the server as the state. The state at time t is represented by $s_t = < \Delta_1, \Delta_2, \dots, \Delta_k, \dots, \Delta_m >$, where $1 \leq k \leq m$. $\Delta_k = |VMQ_k|$ is the number of tasks waiting to be executed in the buffer queue VMQ_k of VM VM_k at time t. For example, a state $< 5, 3, \dots, 2 >$ indicates that there are five tasks waiting to be executed in the buffer of VMQ_1, three tasks waiting to be executed in the buffer of VMQ_2, and two tasks waiting to be executed in the buffer of VMQ_m.

4.2.2. Action Space

The action space A is a set of executable actions. Selecting an action assigns a task to a VM. Because there are m VMs in a server, we use the VM number k to represent the action ($1 \leq k \leq m$). Therefore, the action space $A = 1, 2, 3, \dots, m$. For example, selecting action $a_t = 2$ at time t means that the current task is assigned to the second VM.

4.2.3. Reward

In Q-learning, after taking an action the agent immediately becomes aware of the new state and can receive or observe signals from the environment. These signals, either reward or punishment values, can help the scheduler to learn the optimal policy over time, gradually improving its performance in future task scheduling. Next, we present the design of the reward function used for the proposed scheduling program.

The total time that a task exists in the system includes its queue waiting time and the task processing time on VM_k

$$Time\_T^j_{VM_k} = w_1 * Time\_W^j_{VM_k} + w_2 * Time\_E^j_{VM_k}$$

where $Time\_W^j_{VM_k} = e^k + \sum\limits_{l \in VMQ_k} len_l / P_k$ is the waiting time of task j in VM_k, e^k is the remaining processing time of the task currently being executed in VM_k, len_l is the data length of the task in VMQ_k, and P_k is the processing speed of VM_k.

Moreover, $Time\_E^j_{VM_k}$ is the time required to execute a task after it is assigned to VM_k, while w_1 and w_2 are the weights of two different times.

We define the reward value r as follows:

$$\mathrm{r} = \begin{cases} 1 & (Time\_T^j_{VM_k} \leq Time\_D^j_{VM_k}) \ \& \ (Time\_T^j_{VM_k} = \min\limits_{l \in VMQ_k} (Time\_T^j_{VM_k})) \\ 0 & (Time\_T^j_{VM_k} \leq Time\_D^j_{VM_k}) \\ -1 & (Time\_T^j_{VM_k} > Time\_D^j_{VM_k}) \end{cases}$$

When taking an action a to assign task j to VM_x, if VM_x can meet the deadline of task j and the sum of the waiting time and execution time of task j on VM_x is the lowest among all m VMs, then a positive reward value is given. If task j only meets the task deadline, no reward value is given. If the task deadline cannot be met, a penalty is given. We designed a suitable reward function to train the scheduler with the goal of improving task processing speed and minimizing the makespans and processing times of all tasks.

To schedule tasks reasonably in the server, we propose a task scheduling algorithm called UQRL. We built an RL model based on UCB Q-Learning [42], with the workflow presented below in Algorithm 2.

Algorithm 2: UQRL Task scheduling algorithm

```
1  Initialize Parameters: $\varepsilon, \delta, \gamma$
2      $Q_{(s,a)}, \widehat{Q}_{(s,a)} \leftarrow 0, N_{(s,a)} \leftarrow 0, \forall s \in S, a \in A$
3      $\varepsilon_1 \leftarrow \frac{\varepsilon}{24RMln\frac{1}{1-\gamma}}, H \leftarrow \frac{ln1/((1-\gamma)\varepsilon_1)}{ln1/\gamma}$
4      $\iota(k) = ln(SA(k+1)(k+2))/\delta, \alpha_k = \frac{H+1}{H+k}$
5  for $t = 0, 1, 2, ...$ do
6      Given the current state $s_t$
7      Take action $a_t \leftarrow \arg\max_{a'} \widehat{Q}_{(s_t,a')}$
8      Receive reward $r_t$ and transit to $s_{t+1}$
9      $N_{(s_t,a_t)} \leftarrow N_{(s_t,a_t)} + 1$
10     $k \leftarrow N_{(s_t,a_t)}, b_k \leftarrow \frac{c}{1-\gamma}\sqrt{\frac{H\iota(k)}{k}}$
11     $\widehat{V}_{(}s_{t+1}) \leftarrow \max_{a \in A} \widehat{Q}_{(s_{t+1},a)}$
12     $Q_{(s_t,a_t)} \leftarrow (1-\alpha_k)Q_{(s_t,a_t)} + \alpha_k[r_{(s_t,a_t)}) + b_k + \gamma\widehat{V}_{(s_{t+1})}]$
13     $\widehat{Q}_{(s_t,a_t)} \leftarrow \min(\widehat{Q}_{(s_t,a_t)}, Q_{(s_t,a_t)})$
```

In the algorithm, ε is a relaxation factor that allows the algorithm to deviate from the local optimal solution to an extent in order to obtain a better global solution, δ represents the magnitude of change or updating of the value function during each iteration, γ is the discount factor for rewards, and H is the number of steps per episode.

The goals of the UQRL algorithm are to select an action that has achieved higher rewards than other actions in past explorations and to select an action that has been explored less than other actions. In line 7 of the algorithm, the action with the maximum reward is selected and $\widehat{Q}_{(s,a)}$ is the historical minimum value of the Q function. In line 12 of the algorithm, the b_k term affects the Q value based on the number of times the action has been executed.

For each task in the SQ, the UQRL algorithm is executed to assign it to a VM until all tasks in the SQ are assigned. Considering the current state s_t, the action with the maximum reward is selected and then the state is changed to s_{t+1}. There are m VMs in the server, and the current state is represented as $s_t = (|VMQ_1|, |VMQ_2|, \ldots, |VMQ_k|, \ldots, |VMQ_m|)$. Assuming that the selected action is $a_t = k$, according to step 8 of the algorithm, the next state is $s_{t+1} = (|VMQ_1|, |VMQ_2|, \ldots, |VMQ_k| + 1, \ldots, |VMQ_m|)$.

Each VMQ has a queue capacity c, indicating that a maximum of tasks c can be assigned to the VM. The value can be set based on the actual state of the cloud environment and the total number of tasks. The size of the c value affects the dimension of the state space. Different queue capacities can be set for each VM, or they can be set to the same capacity. Before executing each action, it is first necessary to check whether the current queue capacity c of the VM is full. If it is full, the task is assigned to other VMs. If all VMQs in the server are full, the scheduler waits until there is a VMQ with available capacity before performing task assignment.

Here, the number of states in Q-Learning is denoted as S and the number of available actions in each state is denoted as a. The time complexity of the Q-Learning algorithm is O(S*a*N), where N represents the number of iterations required for convergence. When comparing the UCB strategy with the greedy strategy, the UCB strategy requires fewer iterations to converge, resulting in a lower time complexity. The space complexity of the Q-Learning algorithm is O(S*a), primarily due to the memory space occupied by the Q-table. Neither the Q-Learning algorithms using the UCB strategy nor the greedy strategy alters the size of the Q-table; therefore, they have the same space complexity.

5. Experimental Results

To verify the performance of the proposed QMTSF, we simulated task scheduling in cloud computing on the CloudSim simulation platform. In the simulation experiment, we validated scenarios with 100, 300, 500, 800, and 1000 tasks and with randomly generated task lengths between 500 and 3000. There were four servers (i.e., task categories) in the cloud environment, and each server had five VMs with processing speeds randomly generated between 500 and 1000. To ensure the accuracy of the experiment, we took the average of multiple experimental results as the final experimental result. The average number of experiments conducted in this paper was ten times. We found that when the number of experiments exceeds ten, increasing the number of experiments further had a minimal impact on the average results.

Our experimental process consisted of two scenarios. In the first scenario, we used QMTSF priority task scheduling and compared it with RR and random task scheduling in the first stage of task allocation. In the second stage, we used the UQRL algorithm for scheduling to verify the effectiveness of QMTSF priority task scheduling. In the second scenario, we used QMTSF task scheduling in the first stage and compared the UQRL algorithm with the PSO heuristic algorithm and ε-Greedy Q-Learning algorithm in the second stage to verify the advantages of UQRL algorithm compared with other task scheduling algorithms. Our simulation results showed that the UQRL algorithm can significantly reduce the makespan and average processing time. Finally, we compared the energy consumption of scheduling methods such as QMTSF, Q-Learning, PSO, RR, and random. Our simulation results demonstrate that QMTSF achieves the highest energy efficiency.

5.1. QMTSF Comparison with RR and RANDOM

RR scheduling is a fair scheduling method that assigns tasks in GQ to each server in order. Random assignment, on the other hand, randomly assigns tasks to each server. A performance comparison of these two methods is shown in the figures below.

From Figure 3, it can be observed that when the task number is small, QMTSF has a slightly better makespan compared to RR and Random, though the advantage is not very significant. However, as the number of tasks increases, the makespan of QMTSF increases only slowly, while RR and random experience a noticeable increase in time. Random assignment performs the worst, resulting in a large number of tasks failing to meet their deadlines. From Figure 4, it can be seen that QMTSF maintains a consistently uniform average task processing time as the number of tasks increases, indicating that the task processing time in QMTSF is only dependent on the VM processing speed and task data length, with minimal impact from the task number. While the average task processing time for RR increases with the number of tasks, this increase is within a reasonable range, while random assignment exhibits a significantly higher increase in the average task processing time.

This is mainly because the objective of the RR policy is to balance the load of all VMs. As the number of tasks increases, achieving a balanced task allocation becomes increasingly difficult. On the other hand, the Random policy schedules tasks to servers randomly without considering the current workload of the servers, which can lead to some servers being idle while others are congested with excessive queued tasks. In contrast, QMTSF takes into account both the characteristics of tasks and the performance of servers when allocating tasks. Additionally, it considers task deadlines, which further influences the acquisition of reward values in the UQRL algorithm. RR and random assignment do not consider this aspect, which prevents them from obtaining more optimized Q-values when constructing the Q-table in the second stage.

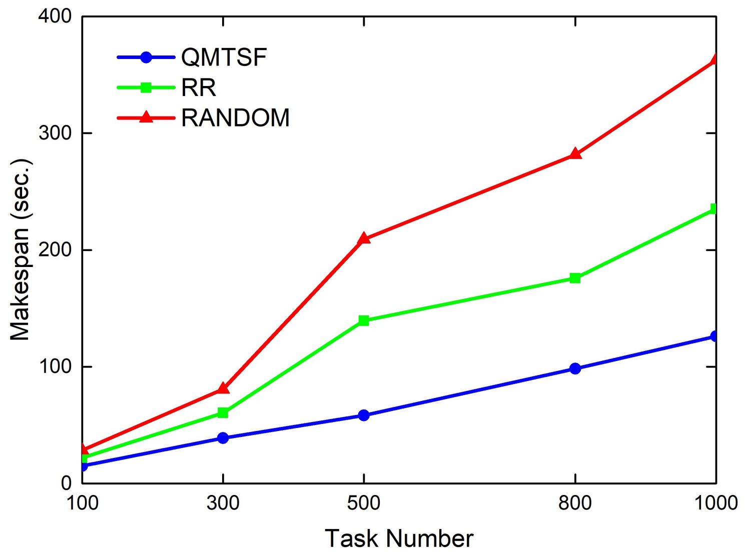

Figure 3. Makespan comparison: different task allocation policies employed in Stage I.

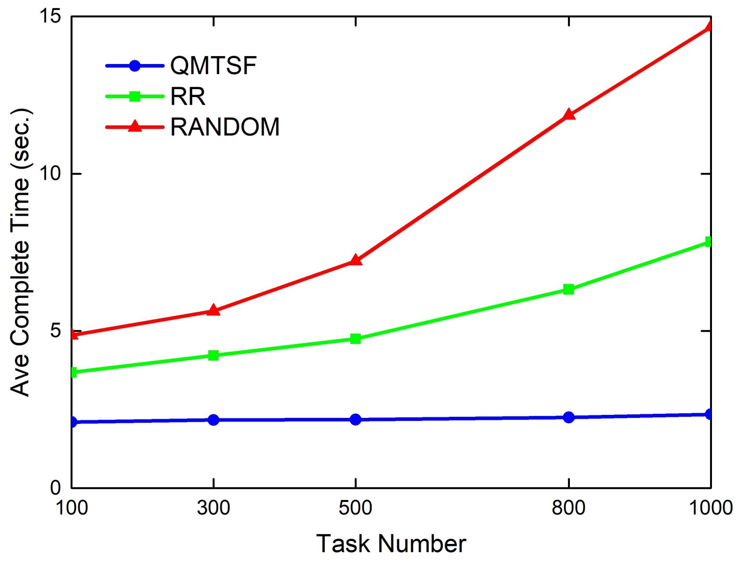

Figure 4. Average processing time comparison: different task allocation policies employed in Stage I.

5.2. UQRL Comparison with Q-Learning and PSO

PSO is a typical heuristic algorithm that is widely applied in various domains, including task scheduling scenarios. In the first stage, we utilized the task allocation method from QMTSF; in the second stage, described here, we employed both UQRL algorithm, Q-Learning algorithm, and PSO algorithm for scheduling to compare their performance.

From Figure 5, it can be observed that when the number of tasks is small, the makespan of PSO is slightly better than UQRL and Q-Learning, with a negligible difference. However, the makespan of UQRL and Q-Learning becomes significantly lower than PSO as the number of tasks increases. The makespan achieved with UQRL is about 20% lower than with PSO. This is because PSO requires multiple iterations to converge during each scheduling process, which incurs additional iteration time. On the other hand, UQRL and Q-Learning can quickly make scheduling decisions after Q-table training is completed. Another approach is to train the algorithms offline and perform task scheduling online, which improves the initial execution efficiency of the algorithms. Compared to Q-Learning, UQRL can realize time savings of approximately 7% thanks to the faster convergence of its UCB algorithm in finding the optimal solution as compared to the greedy algorithm. In this case, PSO, UQRL, and Q-Learning all show a gradual increase in makespan without the significant degradation in performance observed in Figure 3 with random scheduling. This is because dynamic task allocation of QMTSF was already employed in the first stage to achieve optimal matching of tasks to servers. Thus, the performance comparison in this stage focuses on the scheduling of tasks to VMs within the servers.

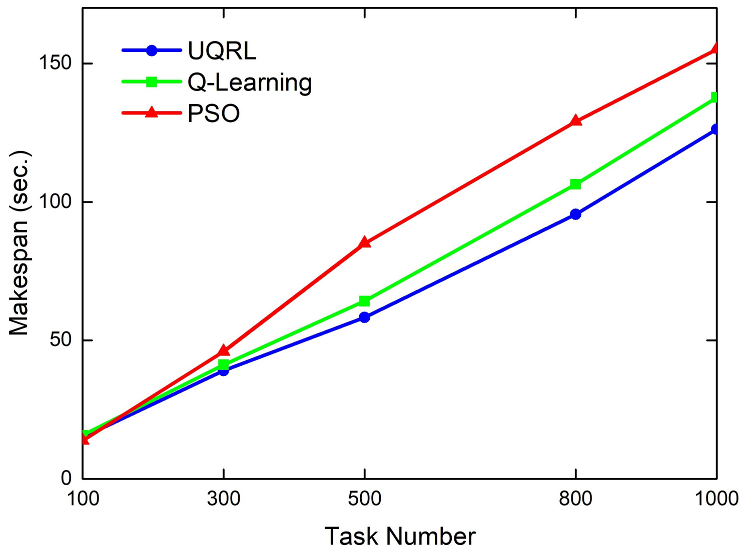

Figure 5. Makespan comparison: different task scheduling policies employed in Stage II.

From Figure 6, it can be observed that UQRL, Q-Learning, and PSO all show relatively stable average task processing times without significant fluctuations as the number of tasks increases. Compared to the PSO algorithm, the UQRL algorithm can reduce the average task processing time by around 15%. This is because as the task number increases, the time consumed by the multiple iterations needed for convergence of the PSO algorithm is averaged among tasks, resulting in a diminishing impact. On the other hand, after the convergence of the Q-table the UQRL algorithm does not incur significant delays in the process of scheduling tasks to specific VMs. The average task processing time of the UQRL algorithm is not significantly different from the Q-Learning algorithm, as both algorithms adopt the same scheduling selection method following convergence of the Q-table.

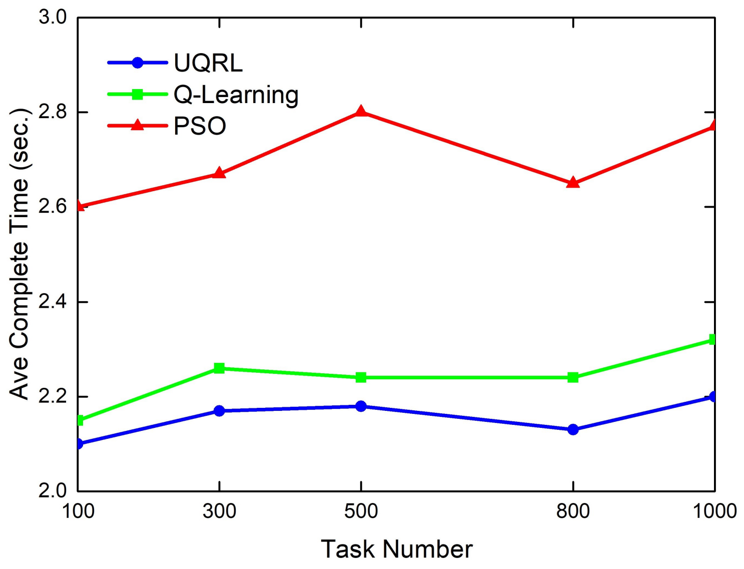

Figure 6. Average processing time comparison: different task scheduling policies employed in Stage II.

5.3. Comparison of CPU Utilization between QMTSF, Q-Learning, PSO, RR, and Random

After validating the performance advantages of QMTSF in terms of the makespan and average task processing time, we further compared the CPU utilization for the different scheduling methods.

From Figure 7, it can be observed that as the task number increases, the CPU utilization of the various scheduling algorithms increases as well. QMTSF exhibits the highest CPU utilization, which, combined with its excellent performance in task execution time, indicates that this scheduling method can quickly complete tasks within a short time and occupy the CPU for a relatively shorter duration. This approach can simultaneously reduce the overall runtime of the entire cloud data center, allowing for more idle time compared to the other scheduling algorithms. Because the power consumption during CPU operation is much higher than during idle time, this can help to achieve the goal of reducing the overall power consumption of the cloud data center. The CPU utilization of the Q-Learning algorithm is second only to QMTSF. Additionally, the figure shows that the PSO scheduling algorithm outperforms RR scheduling, while random scheduling performs the worst. This indirectly confirms the results obtained in the previous experiments regarding task execution time.

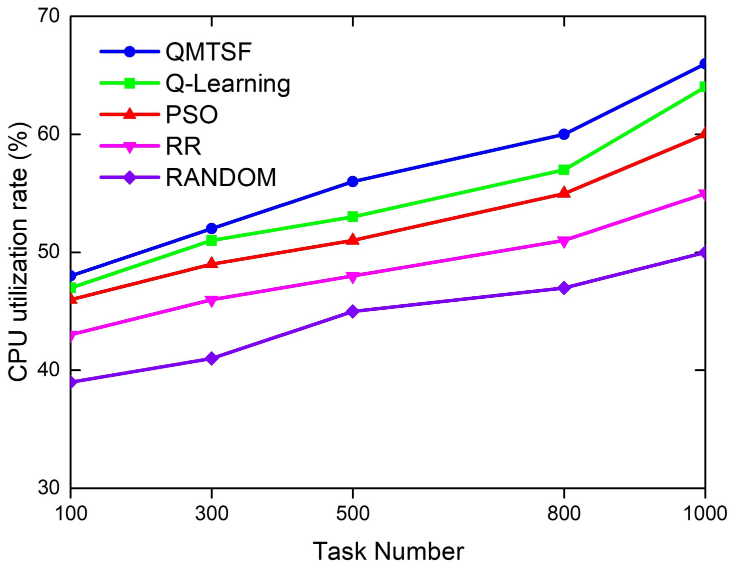

Figure 7. CPU utilization: comparison of different scheduling methods.

Based on the above experimental results, it can be observed that when compared to RR, Random, PSO, and Q-Learning, the QMTSF framework has significant advantages in terms of both makespan and average task processing time. Furthermore, it leads to an improvement in CPU utilization and a reduction in the overall energy consumption of the cloud data center. In particularly, as the number of tasks increases, the performance improvements realized by QMTSF compared to traditional scheduling algorithms such as RR and random assignment become very evident.

6. Conclusions

In this paper, we propose a reinforcement learning task scheduling framework called QMTSF based on the UCB strategy for the task scheduling problem in cloud computing. When tasks are allocated in the cloud data center, they are first assigned to servers that are more proficient in handling certain types of task based on the task type. In task scheduling on servers, the tasks in the queue are sorted by their respective deadlines, then the UQRL algorithm is used to schedule each task in the queue and allocate it to a VM. Using the UQRL algorithm minimizes the makespan while ensuring that task deadlines are reached, reducing the average task processing time, and improving CPU utilization, thereby lowering the overall energy consumption of the cloud data center. Our experimental results demonstrate that the QMTSF method performs better than other task scheduling strategies.

In the future, we plan to apply this framework to task scheduling scenarios involving unmanned aerial vehicles (UAVs). For this, it will be necessary to explore ways of better classifying and mapping tasks while considering their varying resource requirements, duration, and service quality in order to validate the effectiveness of this framework in real-world environments. Additionally, it will be necessary to address the implementation challenges of cloud clusters in scenarios with larger-scale servers within the cloud environment. In addition, we intend to further explore the application of reinforcement learning in multi-objective task scheduling problems.

Author Contributions: Conceptualization, Y.W.; methodology, Y.W. and S.D.; software, Y.W. and W.F.; validation, Y.W.; formal analysis, Y.W.; investigation, S.D.; resources, S.D.; data curation, Y.W. and S.D.; writing—original draft preparation, Y.W.; writing—review and editing, S.D.; visualization, W.F.; supervision, W.F.; funding acquisition, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Natural Science Foundation of China (Grant Nos. 62102196, 62102195, 62172291), Natural Science Foundation of Jiangsu Province (No. BK20200753), Jiangsu Postdoctoral Science Foundation Funded Project (No. 2021K096A), and Postgraduate Research and Practice Innovation Program of Jiangsu Province (KYCX23_1087).

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

Acknowledgments: The authors would like to thank the anonymous reviewers and the editor for their careful reviews and constructive suggestions to help us improve the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Verma, A.; Kaushal, S. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Comput.* **2017**, *62*, 1–19. [CrossRef]
2. Zhang, H.; He, G. An adaptive task scheduling system based on real-time clustering and NetFlow prediction. In Proceedings of the IEEE 5th International Conference on Computer and Communications (ICCC), Chengdu, China, 6–9 December 2019; pp. 77–80.
3. Pinciroli, R.; Ali, A.; Yan, F.; Smirni, E. Cedule+: Resource management for burstable cloud instances using predictive analytics. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 945–957. [CrossRef]
4. Jamil, B.; Shojafar, M.; Ahmed, I.; Ullah, A.; Munir, K.; Ijaz, H. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5581. [CrossRef]
5. Shafi, U.; Shah, M.A.; Wahid, A.; Abbasi, K.; Javaid, Q.; Asghar, M.N.; Haider, M. A novel amended dynamic round robin scheduling algorithm for timeshared systems. *Int. Arab J. Inf. Technol.* **2020**, *17*, 90–98. [CrossRef] [PubMed]
6. Guevara, J.C.; da Fonseca, N.L. Task scheduling in cloud-fog computing systems. *Peer-to-Peer Netw. Appl.* **2021**, *14*, 962–977.
7. Raju, R.; Babukarthik, R.G.; Chandramohan, D.; Dhavachelvan, P.; Vengattaraman, T. Minimizing the makespan using Hybrid algorithm for cloud computing. In Proceedings of the 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, India, 22–23 February 2013; pp. 957–962.
8. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern.* **1996**, *26*, 29–41.
9. Khalili, A.; Babamir, S.M. Makespan improvement of PSO-based dynamic scheduling in cloud environment. In Proceedings of the 23rd Iranian Conference on Electrical Engineering, Tehran, Iran, 10–14 May 2015; pp. 613–618.
10. Eberhart, R.; Kennedy, J. A new optimizer using particle swarm theory. In Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95), Nagoya, Japan, 4–6 October 1995; pp. 39–43.
11. Gabi, D.; Ismail, A.S.; Dankolo, N.M. Minimized makespan based improved cat swarm optimization for efficient task scheduling in cloud datacenter. In Proceedings of the 3rd High Performance Computing and Cluster Technologies Conference, Guangzhou, China, 22–24 June 2019 ; pp. 16–20.
12. Sharma, M.; Garg, R. HIGA: Harmony-inspired genetic algorithm for rack-aware energy-efficient task scheduling in cloud data centers. *Eng. Sci. Technol. Int. J.* **2020**, *23*, 211–224. [CrossRef]
13. Holland, J.H. *Adaptation in Natural and Artificial Systems*; The MIT Press: Ann Arbor, MI, USA, 1975.
14. Wu, Z.; Liu, X.; Ni, Z.; Yuan, D.; Yang, Y. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *J. Supercomput.* **2013**, *63*, 256–293.

15. Meshkati, J.; Safi-Esfahani, F. Energy-aware resource utilization based on particle swarm optimization and artificial bee colony algorithms in cloud computing. *J. Supercomput.* **2019**, *75*, 2455–2496. [CrossRef]
16. Song, A.; Chen, W.N.; Luo, X.; Zhan, Z.H.; Zhang, J. Scheduling workflows with composite tasks: A nested particle swarm optimization approach. *IEEE Trans. Serv. Comput.* **2020**, *15*, 1074–1088. [CrossRef]
17. Huang, X.; Li, C.; Chen, H.; An, D. Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. *Clust. Comput.* **2020**, *23*, 1137–1147. [CrossRef]
18. Zhou, Z.; Li, F.; Zhu, H.; Xie, H.; Abawajy, J.H.; Chowdhury, M.U. An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. *Neural Comput. Appl.* **2020**, *32*, 1531–1541. [CrossRef]
19. Sulaiman, M.; Halim, Z.; Lebbah, M.; Waqas, M.; Tu, S. An evolutionary computing-based efficient hybrid task scheduling approach for heterogeneous computing environment. *J. Grid Comput.* **2021**, *19*, 11. [CrossRef]
20. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
21. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
22. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
23. Wei, Z.; Zhang, Y.; Xu, X.; Shi, L.; Feng, L. A task scheduling algorithm based on Q-learning and shared value function for WSNs. *Comput. Netw.* **2017**, *126*, 141–149. [CrossRef]
24. Khan, M.I.; Xia, K.; Ali, A.; Aslam, N. Energy-aware task scheduling by a true online reinforcement learning in wireless sensor networks. *Int. J. Sens. Netw.* **2017**, *25*, 244–258. [CrossRef]
25. Wei, Z.; Liu, F.; Zhang, Y.; Xu, J.; Ji, J.; Lyu, Z. A Q-learning algorithm for task scheduling based on improved SVM in wireless sensor networks. *Comput. Netw.* **2019**, *161*, 138–149. [CrossRef]
26. Li, F.; Song, X.; Chen, H.; Li, X.; Wang, Y. Hierarchical routing for vehicular ad hoc networks via reinforcement learning. *IEEE Trans. Veh. Technol.* **2018**, *68*, 1852–1865. [CrossRef]
27. Ding, D.; Fan, X.; Zhao, Y.; Kang, K.; Yin, Q.; Zeng, J. Q-learning based dynamic task scheduling for energy-efficient cloud computing. *Future Gener. Comput. Syst.* **2020**, *108*, 361–371. [CrossRef]
28. De Ath, G.; Everson, R.M.; Rahat, A.A.; Fieldsend, J.E. Greed is good: Exploration and exploitation trade-offs in Bayesian optimisation. *ACM Trans. Evol. Learn. Optim.* **2021**, *1*, 1–22. [CrossRef]
29. Ge, J.; Liu, B.; Wang, T.; Yang, Q.; Liu, A.; Li, A. Q–learning based flexible task scheduling in a global view for the Internet of Things. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4111.
30. Liu, J.; Wang, Q.; He, C.; JaffrÃĺs-Runser, K.; Xu, Y.; Li, Z.; Xu, Y. QMR: Q-learning based multi-objective optimization routing protocol for flying ad hoc networks. *Comput. Commun.* **2020**, *150*, 304–316. [CrossRef]
31. Slivkins, A. Introduction to multi-armed bandits. *Found. Trends® Mach. Learn.* **2019**, *12*, 1–286. [CrossRef]
32. Bubeck, S.; Munos, R.; Stoltz, G. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory, Proceedings of the 20th International Conference, ALT 2009, Porto, Portugal, 3–5 October 2009*; Springer: Berlin/Heidelberg, Germany, 2009.
33. Jin, C.; Allen-Zhu, Z.; Bubeck, S.; Jordan, M.I. Is Q-learning provably efficient? *Adv. Neural Inf. Process. Syst.* **2018**, *31*.
34. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **2002**, *47*, 235–256. [CrossRef]
35. Kocsis, L.; SzepesvÃąri, C. Discounted ucb. In Proceedings of the 2nd PASCAL Challenges Workshop, Venice, Italy, 2006.
36. Saito, K.; Notsu, A.; Ubukata, S.; Honda, K. Performance Investigation of UCB Policy in Q-learning. In Proceedings of the IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 777–780.
37. Bai, Y.; Xie, T.; Jiang, N.; Wang, Y. X. Provably efficient q-learning with low switching cost. *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
38. Bae, J.; Lee, J.; Chong, S. Learning to schedule network resources throughput and delay optimally using Q^+-learning. *IEEE/ACM Trans. Netw.* **2021**, *29*, 750–763.
39. Elsayed, M.; Joda, R.; Abou-Zeid, H.; Atawia, R.; Sediq, A.B.; Boudreau, G.; Erol-Kantarci, M. Reinforcement learning based energy-efficient component carrier activation-deactivation in 5G. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021.
40. Li, Z.; Pan, S.; Qin, Y. Multiuser Scheduling Algorithm for 5 G IoT Systems Based on Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2022**, *72*, 4643–4653.
41. Zhao, Y.; Lee, J.; Chen, W. Q-greedyUCB: A New Exploration Policy for Adaptive and Resource-efficient Scheduling. *arXiv* **2020**, arXiv:2006.05902.
42. Dong, K.; Wang, Y.; Chen, X.; Wang, L. Q-learning with ucb exploration is sample efficient for infinite-horizon mdp. *arXiv* **2019**, arXiv:1901.09311