Optimal Dynamic Task Scheduling in Heterogeneous Cloud Computing Environment

Wenlong Ni*, Yuhong Zhang†, and Wei Li†
*99 ZiYang Ave, JiangXi Normal University, NanChang, CHINA
†3100 Cleburne St, Texas Southern University, Houston, USA

***Abstract*—Cloud computing (CC), often necessitates dynamic adjustments due to its inherently fluid nature. In this paper, we introduce a novel dynamic task scheduling model that incorporates reward and holding cost considerations, leveraging the Continuous-Time Markov Decision Process (CTMDP) framework in heterogeneous CC systems. The primary goal of this model is to maximize the overall system reward for the Cloud Service Provider. By solving the Bellman Optimality Equation using the value-iteration method, we can derive an optimal scheduling policy for the dynamic task scheduling model. Additionally, to enhance its practicality in real-world scenarios, we incorporate a model-free reinforcement learning algorithm to obtain the optimal policy for our proposed model without requiring explicit knowledge of the system environment. Simulation results demonstrate that our proposed model outperforms two common static scheduling methods.**

***Index Terms*—Cloud Data Center, Optimal Policy, Cost Function, Q-Learning.**

I. INTRODUCTION

Cloud computing (CC) system provides users with safe and reliable computing resources via the Internet [1]–[6]. The infrastructure of a CC system includes a large-scale, heterogeneous array of servers that vary in performance and resource consumption [7]–[9]. By leveraging virtualization and other technologies, it transforms these physical resources into pools of virtual resources, facilitating the management of computing resources such as CPUs, memory, and storage. As Big Data technology becomes increasingly prevalent, user demands for CC systems are growing increasingly intricate and diverse, making efficient task scheduling a crucial aspect of CC systems, as highlighted in [10], [11].

In recent years, the importance of efficient task scheduling in CC systems has become increasingly apparent. As the demand for cloud services continues to grow, so does the complexity of managing and optimizing resource allocation. To address this challenge, numerous studies have been conducted on task scheduling algorithms for CC systems, aiming to improve performance, hardware utilization, and overall system efficiency. [12] introduces a cloud task scheduling algorithm based on Three Queues, which aims to enhance hardware utilization by effectively managing the distribution of tasks across available resources. This approach helps balance workload and resource utilization, leading to improved overall system performance.

Another approach to CC task scheduling is the use of the ant colony algorithm, as demonstrated in [13]. This method mimics the natural behavior of ants searching for food, where they collaborate to find the shortest path to a food source. Similarly, in task scheduling, the ant colony algorithm enables tasks to be allocated to the most suitable resources based on historical data and optimization techniques. The resulting optimized solution improves system performance by reducing response time and enhancing task completion rates. Furthermore, a min-max task scheduling method is proposed in [14]. This method allocates tasks to achieve better system load balancing and minimize response time. By considering both the maximum and minimum loads across different resources, the algorithm ensures that tasks are distributed evenly, preventing any single resource from becoming overloaded. This approach not only improves system stability but also enhances the overall user experience by providing consistent and responsive service.

These existing studies often focus on enhancing response time performance and load balancing [12]–[15]. Additionally, one study presents a dynamic control model to optimize cloud resource consumption and quality of service [16]. In this model, servers in a CC system are configured as $M/M/1$ queuing systems, treating task arrivals and departures as a Poisson process. Figure 1 illustrates the dynamic task scheduling model, where the dispatcher assigns multiple Poisson streams (tasks) with varying intensities to heterogeneous servers for dynamic scheduling control.

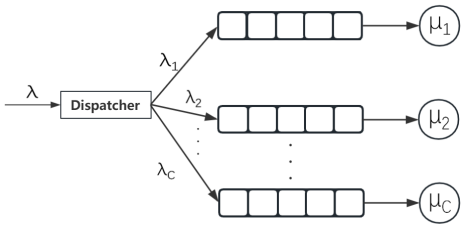

Fig 1. A task scheduling model in [16].

In this paper, we propose a dynamic task scheduling model that incorporates both reward and holding costs. This model is designed to mirror the dynamic nature of cloud computing environments, where tasks are constantly being submitted, processed, and completed. To achieve this, we formulate the task scheduling problem in heterogeneous CC systems

This material is partially based upon work supported by the NSF under Grant Nos. 2302469 and 2318662, and by NASA under Grant No. 80NSSC22KM0052 to Wei Li; and by JiangXi Education Department under Grant No. GJJ191688 to Wenlong Ni.

as a Continuous-Time Markov Decision Process (CTMDP). This approach enables us to optimize the dynamic scheduling of cloud tasks by leveraging real-time performance metrics, rewards, and holding costs associated with heterogeneous servers. To elaborate, the dynamic nature of cloud computing environments poses significant challenges in terms of task scheduling. Heterogeneous servers, varying workload patterns, and dynamic resource availability all contribute to the complexity of the scheduling problem. By modeling the task scheduling problem as a CTMDP, we are able to capture these dynamics and make informed scheduling decisions that maximize the overall reward of the system.

The core objective of our model is to maximize the overall reward of the system by making intelligent scheduling decisions. This is achieved by sensing the performance characteristics, reward structures, and holding costs of various servers in real-time. Other than the traditional Value Iteration (VI) method, as depicted in Figure 2, through reinforcement learning process such as Q-Learning our proposed model allows tasks accepted by the system to be dynamically scheduled by the real-time scheduler to the most suitable server based on the current system state, rather than relying solely on traditional task flow control mechanisms.

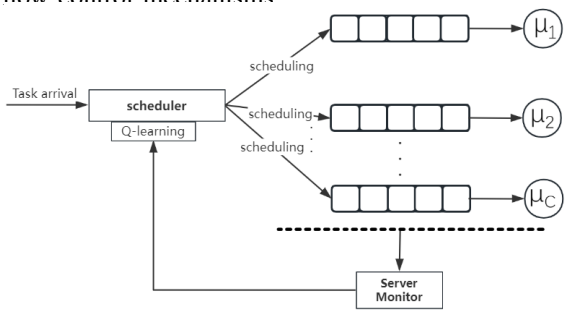

Fig 2. The dynamic task scheduling model.

Our contributions in this paper can be summarized as follows:

- The real-time scheduler within the system plays a pivotal role in efficiently managing the incoming tasks. It meticulously assigns each task to an appropriate server, taking into account the current system state. This process ensures that tasks are processed in an optimal manner, leveraging the available resources and minimizing any potential delays. To achieve this, in this paper we propose a discounted reward model which transforms the optimization problem into a Continuous-Time Markov Decision Process (CTMDP). This transformation enables the scheduler to perceive the dynamics of task arrivals and departures, providing a comprehensive understanding of the system's workload patterns. Additionally, it allows the scheduler to factor in the cost consumption of the system, ensuring that resource allocation is not only efficient but also economically viable. The discounted reward model employed by the scheduler is a crucial component of this optimization process. It allows the system to assign different weights to rewards obtained at different time steps, reflecting the importance of immediate versus delayed gratification. This weighting mechanism enables the scheduler to make informed decisions about task assignments, balancing the need to maximize immediate rewards with the longer-term implications of such decisions.
- Theoretically, if one had complete knowledge of the system environment, including the distribution functions for task arrival and departure, the reward function, and the state transition probability, VI could indeed lead to the derivation of the optimal scheduling policy by solving the Bellman Optimality Equation for the dynamic task scheduling model, which is an elusive goal in the realm of dynamic task scheduling. However, the practicalities of real-world scenarios often present significant challenges that render this theoretical ideal unattainable. To address this challenge, we employ the Q-learning algorithm to derive a near-optimal scheduling policy for the model without requiring extensive knowledge of the system environment. This ensures that the model remains an effective solution in real-world scenarios.
- In this paper, we conducted a numerical analysis of the Continuous-Time Markov Decision Process (CTMDP) model and simulation experiments based on the Q-learning method. The Q-learning method learns the optimal policy by interacting with the environment and updating the Q-values based on the observed rewards. In our experiments, we trained the Q-learning agent using a simulated environment and evaluated its performance through simulation experiments. The achieved value of reward and state actions from Q-learning is quite similar to the one derived from the VI method, which proves the correctness of our proposed model. Further, the performance of our proposed model outperforms a popular task scheduling model and achieves higher system reward.

The remainder of the paper is listed as follows. Section II introduces the system model and the CTMDP model. Section III describes our optimization objective. Section IV shows the numerical analysis results; Section V conducts simulation experiments and makes an analysis of the experimental results. The conclusion is presented in Section VI.

II. MODEL ASSUMPTIONS

This section is comprehensively structured into two subsections, each addressing a distinct aspect of the subject matter. The first subsection delves into the system model, elucidating the essential parameters and components that constitute it. This exploration is crucial as it lays the foundation for the subsequent discussions and analysis. The second subsection introducdx the CTMDP model. This model is derived from the descriptions and definitions established in the previous subsection, which is analyzed in depth, highlighting its important components and their role within the broader context.

A. SYSTEM MODEL

1) The number of VMs in the CC system is limited, which is represented as C. Each VM is consider as a server and

is assigned a unique identifier, namely P_i, $i = 1, \ldots, C$. These servers have their own characteristics, some excel at handling computationally intensive tasks, while others are better at handling IO-intensive tasks. Therefore, finding the most suitable server for each task has become an important issue in the CC system. In the CC system, tasks come in randomly. The randomness of tasks poses a significant challenge to the system. Tasks may arrive at any time, and the processing time for each task is unknown. This requires the system to possess a high degree of flexibility and adaptability to ensure efficient task processing without wasting server resources. To achieve this goal, the CC system relies on intelligent task scheduling strategies. These strategies dynamically allocate the most suitable server to each task based on the nature of the task, the status of the server, and the system's load situation.

2) The state of a server engaged in task processing is designated as "busy," whereas a server that is not currently engaged in any task processing is denoted as "idle." Each server is configured as a queuing system with a designated buffer capacity. Tasks assigned to a busy server await processing in the buffer queue. Tasks will not be allocated to a busy server without available buffer capacity for processing. When all the servers in the system are occupied and no buffers have remaining capacity, incoming tasks will be rejected by the system. The maximum number of user tasks that can be accommodated in P_i simultaneously is denoted as N_i, $i = 1, \ldots, C$. The operational state of a server is indicated by the number of tasks staying in the server. Noting the number of tasks staying in P_i as $n_i, i = 1, \ldots, C$, and the operating state of the P_i is

$$\mathrm{O}_i = \begin{cases} \text{idle}, & n_i = 0, \\ \text{busy}, & n_i > 0. \end{cases}$$

3) The arrival of user tasks is deemed as a Poisson process with a rate denoted as λ. In the CC system, the servers exhibit heterogeneity and possess distinct service rates. It is postulated that the duration for P_i to accomplish a task follows an exponential distribution characterized by a rate of μ_i. The collection of service rates for all servers within the system can be represented as $\{\mu_1, \mu_2, \cdots \mu_i, \ldots, \mu_C\}$.

4) The CC system earns rewards by accepting user tasks and providing services. The reward obtained by the system from accepting tasks is defined as a constant R, where $R \geq 0$. As we all know, heterogeneous servers will generate different costs (for example, different hardware power consumption, virtual machine prices, maintenance costs, etc.). Each server in the system has its own cost coefficient, and CSP can determine its cost coefficient based on the historical information of each server in the system.

B. CTMDP MODEL

A CTMDP is a stochastic model that generalizes the discrete-time Markov Decision Process (MDP) to continuous time. It consists of a set of states, a set of actions, transition rates between states, and a reward function that assigns rewards to state-action pairs. The optimization objective in a CTMDP is typically to find a policy that maximizes the expected discounted reward over an infinite horizon. Specifically, let s_t represents the state at time t, a_t represents the action to be taken under state s_t, and $r(s_t, a_t)$ represents the return when action a_t under state s_t, our objective is to find an optimal policy π_α that can bring the maximum total expected discounted reward $v_\alpha^\pi(s)$ as defined below for every initial state s.

$$v_\alpha^\pi(s) = E_s^\pi \left\{ \int_0^\infty e^{-\alpha t} r(s_t, a_t) dt \right\}. \tag{1}$$

Here, a policy π specifies the decision rule to be used at every decision epoch. It gives the decision maker a prescription for action selection for any possible future system state or history.

In this subsection, the optimization problem is modeled as an CTMDP, and some important components of the CTMDP are described.

1) State Space:

In the CTMDP model, the state of the servers and the events occurring in the system (including task arrival and departure) together define the state space of the system. The state of all servers in the system can be represented as $\mathbf{n} = (n_1, n_2, \ldots, n_i, \ldots, n_C)$. For example, $\mathbf{n} = (2, 0, \ldots, 0, \ldots, 0)$ means that there are 2 tasks staying in server P_1 and no tasks staying in any of the other servers. After a user task is processed, the task will leave the system. The arrival event of the user task is denoted as A, and the departure event of the user task as D. A task departure event that occurred as a result of P_i finishing processing a task is denoted as D_i. Based on the above description, the events that may occur in the system can be represented as $\mathbf{e} \in \{A, D_1, D_2, \cdots, D_i, \ldots, D_C\}$.

With all these definitions, the system state space is

$$S = \{s \mid s = \langle (n_1, n_2, \ldots, n_i, \ldots, n_C), \mathbf{e} \rangle = \langle \mathbf{n}, \mathbf{e} \rangle\}.$$

2) Action Space:

When a user task arrives at the system, the system needs to make admission and scheduling decisions based on the current system state. The decisions to be made include whether to accept the task and, if so, to which server the task will be scheduled for processing. The system only makes decisions when those events occur, so we call the points in time of those events epoch. The decision action taken by the system at an epoch is denoted as a.

As mentioned in the system model, The system only rejects a task when there is no capacity remaining in any of the buffers. The action of the system to reject a task is noted as $a = 0$. The system accepts a user task and schedules it to P_i for service, the corresponding action is noted as $a = i$ and $i \in \{1, 2, \ldots, C\}$. When a user task in the system is processed and leaves the system, the system does not need to make an additional action but continues to run the system. The corresponding action is noted as $a = -1$.

Based on the above description, the system action space is defined as Action $= \{-1,\ 0,\ 1, \ldots\ i, \ldots\ C\}$. The possible actions that the system can take in different system states are

$$A(s) = \begin{cases} \{0, 1, \ldots, i, \ldots, C\}, & s = \langle \mathbf{n}, \mathbf{e} \rangle, \mathbf{e} = A, \\ -1, & s = \langle \mathbf{n}, \mathbf{e} \rangle, \mathbf{e} = D. \end{cases}$$

3) Transition Probability:

For the state $s = \langle (n_1, n_2, \ldots, n_i, \ldots, n_C), \mathbf{e} \rangle = \langle \mathbf{n}, \mathbf{e} \rangle$, after the system takes action a, the new state of the system is represented as $s' = \langle \mathbf{n}', \mathbf{e}' \rangle$, and the $q(s' \mid s, a)$ is denoted as the corresponding transition probability. For the arrival event A, if the system takes an action $a = 0$ to reject the task, the operational state of all the servers will not change, which means $\mathbf{n}' = \mathbf{n}$. If the system accepts a task and scheduling it to P_i, the next state $\mathbf{n}'$ is $\mathbf{n}^{\mathrm{i}}$ which is

$$\mathbf{n}^{\mathrm{i}} = (n_1, n_2, \ldots, n_i{+}1, \ldots, n_C).$$

For the departure event D_i, the system takes the action $a = -1$ to continue running the system, $\mathbf{n}'$ is $\mathbf{n}_{\mathrm{i}}$ which is

$$\mathbf{n}_{\mathrm{i}} = (n_1, n_2, \ldots, n_i{-}1, \ldots, n_C),$$

where $n_i > 0$.

Based on the assumptions in the model, it is easy to know that the interval duration between two decision-making epochs of the system follow exponential distribution, and the distribution of time between two epochs is

$$F(t \mid s, a) = 1 - e^{-\beta(s,a)t}, t \geq 0.$$

The state transition probability of the system is

$$q(s' \mid s, a) = \begin{cases} \frac{\lambda}{\beta(s,a)}, & s' = \langle \mathbf{n}', A \rangle, \\ \frac{\mathbb{I}(n_i > 0) \cdot \mu_i}{\beta(s,a)}, & s' = \langle \mathbf{n}', D_i \rangle, n_i \in \mathbf{n}', \end{cases}$$

where $\mathbb{I}$ is the indicator function.

Here $\beta(s, a)$ is the average rate at which the system moves from the current decision epoch to the next after taking action a in state s, and $\beta(s, a)$ equals the summation of the rates of all possible events after taking action a in state s. It is noted that when n_i = 0, P_i does not cause a departure event to occur. Based on these descriptions, $\beta(s, a)$ is

$$\beta(s, a) = \lambda + \sum_{n_k \in \mathbf{n}} \mathbb{I}(n_k > 0) \cdot \mu_k.$$

4) Reward Function:

Let $\tau(s, a)$ be the expected time duration of the current epoch to the next epoch with the current state s taking action a, and $\tau(s, a)$ is

$$\tau(s, a) = \beta(s, a)^{-1}.$$

According to the discount reward model defined in [17], the expected discount reward received by the system at the current decision epoch is denoted as $r(s, a)$, and $r(s, a)$ is

$$\begin{aligned} r(s, a) &= k(s, a) + c(s, a) E_s^a \left\{ \int_0^\tau e^{-\alpha t} dt \right\} \\ &= k(s, a) + c(s, a) E_s^a \left\{ \frac{[1 - e^{-\alpha \tau}]}{\alpha} \right\} \\ &= k(s, a) + \frac{c(s, a)}{\alpha + \beta(s, a)}. \end{aligned}$$

Here α is the time discount factor, and $k(s, a)$ is the immediate reward the system received for taking action a in state s. $c(s, a)$ is the unit time cost incurred by the system from the current decision epoch to the next. Both the reward and the cost of running the system can be set by CSP. To simplify the representation, $k(s, a)$ is

$$k(s, a) = \begin{cases} 0, & \mathbf{e} \in \{D_1, D_2, \ldots, D_C\}, a = -1, \\ 0, & \mathbf{e} = A, a = 0, \\ R, & \mathbf{e} = A, a = i, i \in \{1, 2, \ldots, C\}. \end{cases}$$

Where R is a positive number denoting the reward that the system obtains immediately after accepting a user task.

Furthermore, $c(s, a)$ represents the holding cost rate at state s subsequent to the execution of action a. Let $f(\mathbf{n})$ denote the cost rate when the system is in state s. Then, $c(s, a)$ can be expressed as

$$c(s, a) = \begin{cases} -f(\mathbf{n}^i), & e = A, a = i, i = 1 \ldots C, \\ -f(\mathbf{n}), & e = A, a = 0, \\ -f(\mathbf{n}_i), & e = D_i, a = -1. \end{cases}$$

III. OPTIMIZATION OBJECTIVES

Our optimization objective is to maximize the long-term discounted reward of the system. The VI method and Q-learning algorithm were used to achieve this goal. VI is a dynamic programming approach that relies on the concept of value functions. These functions estimate the expected cumulative reward an agent can achieve by following a particular policy from a given state. VI starts with an initial guess for the value functions and iteratively updates them until convergence. On the other hand, Q-learning directly learns the optimal action-value function, also known as the Q-function, which estimates the expected cumulative reward for taking a particular action in a given state and following the optimal policy thereafter. The Q-function is iteratively updated using the Bellman equation, which captures the relationship between the current state, action, reward, and the next state. The agent selects actions based on the Q-function, gradually learning to favor actions that lead to higher long-term rewards.

A. Value Iteration

For the VI method, let $v(s)$ denote the state value function to evaluate the value of state. According to the Bellman Optimality Equation and the description of the CTMDP model, the objective is that for each state s

$$v(s) = \max_{a \in K(s)} \left\{ r(s, a) + \gamma \cdot \sum_{s' \in S} q(s' \mid s, a) v(s') \right\},$$

where $\gamma = \beta(s, a)/(\alpha + \beta(s, a))$.

For the departure event D_i, according to the Bellman Equation, we have

$$\begin{aligned} v(\langle \mathbf{n}^{\mathrm{i}}, D_i \rangle) = \frac{1}{\alpha + \beta(s, a)} \big[& \lambda v(\langle \mathbf{n}, A \rangle) \\ & + \sum_{k=0}^{C} \mathbb{I}(n_k > 0) \mu_k v(\langle \mathbf{n}, D_k \rangle) \\ & - f(\mathbf{n}) \big]. \end{aligned}$$

Where $n_k \in \mathbf{n}$. Similarly, let $j \in \{1, 2, \ldots, C\}$, $i \neq j$, and from these analyses, it is not too hard to verify that

$$v(\langle \mathbf{n}^{\mathrm{i}}, D_i \rangle) = v(\langle \mathbf{n}^{\mathrm{j}}, D_j \rangle).$$

The above equation shows that for a departure event, $v\left(\left\langle \mathbf{n}^{\mathrm{i}}, D_i\right\rangle\right)$ is only related to the changed operational state of the servers in the system and not to the type of departure event that occurred. Thus, a new function $X(\mathbf{n})$ is defined by us, and $X(\mathbf{n}) = v\left(\left\langle \mathbf{n}^{\mathrm{i}}, D_i\right\rangle\right), i \in \{1, 2, \ldots, C\}$.

For the arrival event, using the example of the system taking the action $a = i$, $i > 0$, it can be obtained that

$$\begin{aligned} v\left(\langle \mathbf{n}, A\rangle, i\right) = R + \frac{1}{\alpha + \beta(s,a)} \Big[& \lambda v\left(\left\langle \mathbf{n}^{\mathrm{i}}, A\right\rangle\right) \\ & + \sum_{k=0}^{C} \mathbb{I}\left(n_k > 0\right) \mu_k v\left(\left\langle \mathbf{n}^{\mathrm{i}}, D_k\right\rangle\right) \\ & - f(\mathbf{n}^{\mathrm{i}})\Big]. \end{aligned}$$

And, for the action of $a = 0$, we have

$$\begin{aligned} v\left(\langle \mathbf{n}, A\rangle, 0\right) = R + \frac{1}{\alpha + \beta(s,a)} \Big[& \lambda v\left(\langle \mathbf{n}, A\rangle\right) \\ & + \sum_{k=0}^{C} \mathbb{I}\left(n_k > 0\right) \mu_k v\left(\langle \mathbf{n}, D_k\rangle\right) \\ & - f(\mathbf{n})\Big]. \end{aligned}$$

Based on the above analysis, $v\left(\langle \mathbf{n}, A\rangle\right)$ is

$$v\left(\langle \mathbf{n}, A\rangle\right) = \begin{cases} X(\mathbf{n}) & , a = 0, \\ X(\mathbf{n}^{\mathrm{i}}) + R & , a = i, i \in \{1, 2, \cdots, C\}. \end{cases}$$

Combined with the Bellman Optimality Equation, the optimal value function for the arrival event occurs can be obtained by the following equation. It should be noted that the example here is that all buffers have remaining capacity.

$$\begin{aligned} & v\left(\langle \mathbf{n}, A\rangle\right) \\ & = \max(R + \max\left[X\left(\mathbf{n}^1\right), X\left(\mathbf{n}^2\right), \cdots, X\left(\mathbf{n}^C\right)\right], X\left(\mathbf{n}\right)). \end{aligned}$$

To apply the VI method to solve the continuous time problem, the optimal equation was uniformized using the rate uniformization technique from Chp 11.5 in [17]. By the theory $Theorem$ 11.3.2 of [17], it can be shown that the VI method will converge to the optimal policy and the optimal scheduling policy for the dynamic task scheduling model is a deterministic policy, which means that the optimal action to be taken for each state is a fixed value.

The $X(\mathbf{n})$ will be solved by the VI method. The pseudo-code of the VI method is shown below.

Remark: Although the verification for the optimal policy to be a threshold policy is not provided in this paper, the following experimental section reveals that the optimal policy behaves as a threshold policy when the cost function is an non-decreasing function. We will continue our research in this direction.

B. The Q-learning

It is well known that the VI method requires complete knowledge of the environment to solve the optimal equation. The Q-learning algorithm can find the optimal policy or near-optimal policy by interacting with data without the knowledge. Let $Q(s, a)$ denote the action-value function to evaluate the value of taking action a in state s. The iterative formulation of the Q-learning algorithm is

$$Q(s,a) = Q(s,a) + \alpha' \cdot \left(r(s,a) + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right),$$

where α' is the learning rate, and γ is a discount factor to ensure convergence. The objective of Q-learning is to find a policy $\pi(s)$ by continuously iterating $Q(s, a)$, and $\pi(s)$ is

$$\pi(s) = \arg \max_{a \in K(s)} Q(s, a), s \in S.$$

The system takes a fixed action $a = -1$ when the departure event D occurs. Therefore, the $Q(s, a)$ that need to be solved is $Q(\left\langle \mathbf{n}^{\mathrm{i}}, A\right\rangle, a)$. By using the above equation and the solved $Q(\left\langle \mathbf{n}^{\mathrm{i}}, A\right\rangle, a)$, the optimal or near-optimal policy $\pi(s)$ can be obtained. The pseudo-code of the Q-learning is as follows.

Algorithm 1: The value-iteration

Data: a small threshold $\theta > 0$ determining accuracy of estimation. Initialize $X(\mathbf{n})$, for all $s \in S$.
Result: $X(\mathbf{n})$ under optimal policy, for all $s \in S$.

```
Δ ← 1;
while Δ > θ do
    for each s ∈ S do
        x ← X(n);
        X(n) ← max_a {r(s,a) + E[γ · v(s')]};
        Δ ← max(Δ, |x − X(n)|);
    end
end
return X(n);
```

Algorithm 2: The Q-learning

Data: total simulation time T, learning rate α', Initialize $Q(s, a)$ for all $s \in S$.
Result: $Q(s, a)$ with approximation to the value of $Q(s, a)_\pi$ for all $s \in S$.

```
current_time = 0;
while current_time < T do
    observe state s.
    choose action a from K(s) by using ε − greedy.
      observer r(s,a), s';
    Q(s,a) ← Q(s,a) + α' ·
      (r(s,a) + γ max_{a'} Q(s',a') − Q(s,a));
    update current_time;
end
return Q(s,a);
```

IV. NUMERICAL ANALYSIS

In this section, the VI method and the Q-learning are used to solve the optimal policy for the CTMDP model, respectively. To show our results conveniently, The number of servers is set C=2, and the maximum queue length of the servers is N_1=N_2=5. It can be easily seen that the solution and analysis

are similar for the case more servers and larger buffers. Some of the necessary parameters are listed in TABLE I. All parameters given are illustrative and can be set according to the specific cloud computing environment.

Parameter	Value	Memo
α	0.5	The time discount factor
λ	10	Task arrival rate
R	3	Immediate reward for accepting a task
μ_1	5	Service rate of P_1
μ_2	10	Service rate of P_2
N_1	5	Maximum queue length of P_1
N_2	5	Maximum queue length of P_2

TABLE I
THE EXPERIMENTS PARAMETERS.

Heterogeneous servers have different cost coefficient, which increases with increasing service rate of the server [16]. Refer to the method for setting the cost function in [16], the cost function is set to $c(s,a) = \mathrm{T}_1 n_1 + \mathrm{T}_2 n_2$, where T_1 and T_2 are the cost coefficients of P_1 and P_2, respectively, $\mathrm{T}_1 \leq \mathrm{T}_2$.

Noting our definition of $v\left(\langle \mathbf{n}, A\rangle\right)$ in the previous section, by solving for the value of the optimal $X(\mathbf{n})$, the theoretical optimal decision policy can be obtained. First, the cost function is set to $c_1(s,a) = n_1 + n_2$. The $X(\mathbf{n})$ for each state obtained by the VI method is shown in TABLE II, and the optimal actions obtained from TABLE II are shown in TABLE III, where 1 and 2 mean scheduling the task to P_1 and P_2 for processing, respectively, and 0 means rejecting the task.

	0			$n_2 \rightarrow$		5
0	55.88	55.67	55.30	54.78	54.10	53.26
	55.61	55.35	54.90	54.29	53.53	52.59
	55.10	54.81	54.28	53.56	52.65	51.53
$n_1 \downarrow$	54.40	54.06	53.46	52.62	51.53	50.17
	53.49	53.12	52.43	51.44	50.13	48.43
5	52.40	51.97	51.16	49.97	48.36	46.18

TABLE II
$X(n)$ WITH $c_1(s,a)$.

	0			$n_2 \rightarrow$		5
0	2	1	1	1	1	1
	2	2	2	1	1	1
	2	2	2	2	2	1
$n_1 \downarrow$	2	2	2	2	2	1
	2	2	2	2	2	1
5	2	2	2	2	2	0

TABLE III
THE OPTIMAL ACTION FOR EVERY STATE WITH $c_1(s,a)$.

From TABLE III, it can be seen that the theoretical optimal policy shows a threshold scheduling policy. With the cost function $c_1(s,a)$, scheduling task to P_2, which has a faster processing rate, for processing is often a better choice. As the load on P_2 increases, scheduling tasks to P_1 will gradually become a better choice. Next, let $c_2(s,a) = n_1 + 2n_2$, and the $X(\mathbf{n})$ for each state can be obtained in the same way. The corresponding optimal actions are shown in TABLE IV.

From the comparison between TABLE III and TABLE IV, it can be seen that the threshold for scheduling tasks to P_1 will be lowered due to the increase in the cost coefficient of P_2. To save some space, with the cost function $c_2(s,a)$, the policy

	0			$n_2 \rightarrow$		5
0	2	1	1	1	1	1
	2	2	1	1	1	1
	2	2	1	1	1	1
$n_1 \downarrow$	2	2	2	1	1	1
	2	2	2	1	1	1
5	2	2	2	2	2	0

TABLE IV
THE OPTIMAL ACTION FOR EVERY STATE WITH $c_2(s,a)$.

obtained by the Q-learning after training for 1×10^6 units of time in a simulation environment is shown in TABLE V.

Note from TABLE IV and TABLE V that the Q-learning effectively converges to a policy that is quite similar to the theoretical optimal policy solved by VI method, and the minor difference between the policy obtained by the Q-learning and the theoretical optimal policy is due to sampling errors caused by the computational accuracy and the number of events in the simulation. Similar results can be obtained for the case under other cost functions.

	0			$n_2 \rightarrow$		5
0	2	1	1	1	1	1
	2	2	1	1	1	1
	2	2	2	1	1	1
$n_1 \downarrow$	2	2	2	1	1	1
	2	2	2	1	1	1
5	2	2	2	2	2	0

TABLE V
THE OPTIMAL ACTIONS OBTAINED BY Q-LEARNING WITH $c_2(s,a)$.

V. CONCLUSION

In this paper, we introduce a novel dynamic task scheduling model tailored for heterogeneous cloud computing systems. To capture the environmental dynamics, we formulate the task scheduling problem as a Continuous-Time Markov Decision Process (CTMDP). With a focus on optimizing system reward by balancing task processing rewards and costs, we employ the Value Iteration (VI) method to derive a theoretically optimal scheduling policy for our model. To enhance practical relevance, we employ the Q-learning algorithm to discover a near-optimal policy in the absence of environmental knowledge, and compare it with the theoretically optimal policy through numerical analysis. Our findings offer CSPs a robust framework for operating Cloud Computing (CC) systems in a more cost-effective manner. Future research directions involve conducting additional simulations across diverse conditions and perspectives to assess the performance of our model in a broader range of scenarios.

REFERENCES

[1] P. K. Senyo, E. Addae, and R. Boateng, "Cloud computing research: A review of research themes, frameworks, methods and future research directions," *International Journal of Information Management*, vol. 38, no. 1, pp. 128–139, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401217305923

[2] A. K. Sandhu, "Big data with cloud computing: Discussions and challenges," *Big Data Mining and Analytics*, vol. 5, no. 1, pp. 32–40, 2022.

[3] M. Singh, "Virtualization in cloud computing- a study," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018, pp. 64–67.

[4] J. Lin, D. Cui, Z. Peng, Q. Li, J. He, and M. Guo, "Virtualized resource scheduling in cloud computing environments: An review," in *2020 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, 2020, pp. 303–308.

[5] I. Odun-Ayo, O. Ajayi, and C. Okereke, "Virtualization in cloud computing: Developments and trends," in *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, 2017, pp. 24–28.

[6] J. K. Meena and R. Kumar Banyal, "Efficient virtualization in cloud computing," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 227–232.

[7] K. Li, "Optimal power and performance management for heterogeneous and arbitrary cloud servers," *IEEE Access*, vol. 7, pp. 5071–5084, 2019.

[8] J. Mars, L. Tang, and R. Hundt, "Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity," *IEEE Computer Architecture Letters*, vol. 10, no. 2, pp. 29–32, 2011.

[9] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157401372300062X

[10] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17321519

[11] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, p. 100841, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221065022100002X

[12] Y. Yu and Y. Su, "Cloud task scheduling algorithm based on three queues and dynamic priority," in *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, 2019, pp. 278–282.

[13] H. Liu, "Research on cloud computing adaptive task scheduling based on ant colony algorithm," *Optik*, vol. 258, p. 168677, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402622000936

[14] D. Gnanaprakasam, M. Mohanraj, T. A. S. Srinivas, S. Bhaggiaraj, B. J, and S. Sivankalai, "Efficient task scheduling in cloud environment based on hyper min max task scheduling," in *2023 International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, 2023, pp. 1–6.

[15] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: Adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/3/920

[16] W. Bai, J. Zhu, S. Huang, and H. Zhang, "A queue waiting cost-aware control model for large scale heterogeneous cloud datacenter," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 849–862, 2022.

[17] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.