Deep Reinforcement Learning-based DAG Task Scheduling Algorithm for Cloud Computing

Jiaxin Su

Abstract—**As the paradigm of cloud computing burgeons, task scheduling emerges as a critical mechanism in enhancing resource utilization and quality of service. However, scheduling tasks modeled by Directed Acyclic Graphs (DAGs) poses significant challenges due to their inherent parallelism and dependency constraints. Deep reinforcement learning (DRL) offers a promising approach to address these challenges, yet existing approaches such as the Deep Deterministic Policy Gradient (DDPG) algorithm are hindered by issues including unstable Actor network training. This study introduces an advanced DDPG-based scheduling algorithm tailored for the unique complexities of DAG cloud computing task scheduling. We fortified the Actor network by incorporating a supervised learning approach to adjust its loss function, thereby stabilizing training outcomes. Empirical analyses across several critical metrics—task completion time, server energy consumption, and standard deviation of resource utilization—demonstrate that our refined model substantially outperforms traditional scheduling methods and unmodified DRL algorithms. The findings affirm that the enhanced DDPG algorithm not only expedites scheduling effectiveness but also reduces energy consumption while maintaining service quality, showcasing the extensive potential and pragmatic value of applying DRL in cloud computing task scheduling. Our improved approach is poised to contribute a fresh perspective to future research and application in DAG task scheduling.**

Index Terms—**DAG, DRL, DDPG, Cloud Computing, Task Scheduling.**

I. Introduction

With the rapid development of distributed computing, task scheduling represented by directed acyclic graph (DAG) has become a typical challenge in cloud computing. Different from non-DAG tasks that usually contain linear independent execution sequences, DAG tasks are characterized by complex operations, multiple dependencies, and parallel execution paths. These remarkable features inevitably lead to a series of scheduling challenges in practical application scenarios, such as coordinating data dependencies, minimizing execution time, and optimizing resource allocation[1], which motivates this research.

Contemporary Scheduling Solutions Some of these approaches focus on static offline batch jobs, which are significantly deficient for dynamically changing workloads in cloud environments. In contrast to offline methods, online scheduling techniques usually employ heuristic algorithms which are significantly less efficient when faced with a large number of dynamically changing tasks. The main reason for this is that these approaches suffer from significant shortcomings when dealing with multiple simultaneous task submissions, which are usually processed in a sequential manner without taking into account the interconnectedness of the tasks. In addition, the high degree of heterogeneity and unpredictability inherent in real-world cloud tasks is often overlooked[2].

Therefore, our research introduces a Deep Reinforcement Learning (DRL)-based task scheduling framework, an approach that promises to revolutionize the dynamics of task allocation. Reinforcement Learning (RL) is a component of machine learning that enhances an agent's ability to derive optimal policies in large and complex state interactions, while DRL mitigates this dilemma by integrating deep learning models to moderate the challenges posed by overly large high-dimensional spaces[3]. We formulate online task scheduling as a constrained dynamic optimization puzzle and address it through a DDPG network designed to find optimal task allocation schemes. The network is adept at learning directly from empirical data without a priori knowledge and can make informed scheduling decisions based on iteratively received VM requests[4]. Specifically, the library of task and VM information received by the scheduler is parsed into the state of the agent, while the response time of the task is the pivot of its action reward[5].

Through this exploration, we fill a gap in contemporary research by boldly breaking down the complexity of DAG task scheduling and constructing a robust, adaptable framework that not only withstands the unpredictable changes in cloud computing operations, but also significantly improves efficiency and effectiveness.

II. Related Work

A. Deep Reinforcement Learning

Reinforcement Learning (RL) is an important branch in the field of machine learning, which aims to perceive information from the environment through an intelligent body agent (Agent) and map a set of actions accordingly[6]. Reinforcement learning aims to allow intelligent agents to make decisions in the environment, with the goal of learning through interactions in order to achieve specific goals or maximize cumulative rewards[7]. Distinguished from supervised and unsupervised learning, reinforcement learning focuses on learning without explicit labeling. The method is based on four elements: intelligences, states, actions and rewards[8], by performing actions in the environment and receiving rewards or punishments. Through this process, the intelligent body gradually learns how to choose actions to optimize long-term gains[9].

Manuscript received June 01, 2024
JiaXin Su, School of computer science and technology, Tiangong University, Tianjin, China.

The essence of reinforcement learning is to iteratively solve the optimal policy using Bellman's equation in the framework of Markov Decision Process (MDP)[10,11]. By continuously updating the policy and the value function, the reinforcement learning algorithm gradually approaches the optimal solution, enabling the intelligence to make optimal decisions in different states.

Deep Reinforcement Learning (DRL) is a machine learning technique that integrates the perceptual capability of deep learning and the decision-making capability of reinforcement learning[12]. As shown in Figure 1 in DRL the Agent is the central role, which decides the Action to be taken by observing and analyzing the current State of the Environment. For each action performed, the Environment provides the Agent with a Reward or Penalty to evaluate the effectiveness of the action taken. The goal of the intelligent body is to learn the optimal action strategy through continuous interaction with the environment, i.e., to choose those actions that maximize the cumulative long-term reward. This cyclic process continues, and by continuously optimizing the experience gathered, the intelligent body refines its understanding of the environment and its coping strategies until it is able to make effective decisions in a variety of states[13].

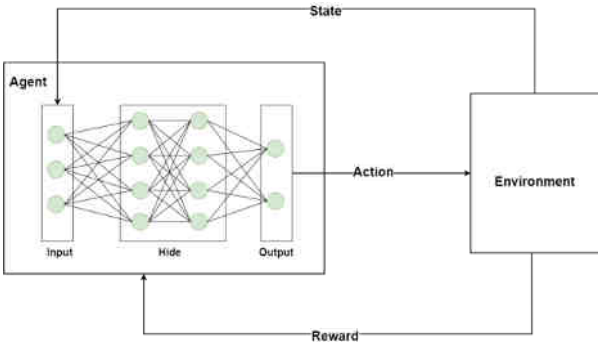


Figure 1: Deep Reinforcement Learning Architecture

B. *Deep Deterministic Policy Gradient Algorithm*

The Deep Deterministic Policy Gradient (DDPG) algorithm is a reinforcement learning algorithm proposed in the context of combining deep learning and policy gradient methods, and is particularly suitable for problems with continuous action spaces. DDPG was proposed by Lillicrap et al[14] in 2015, which introduces deep learning techniques in the domain of policy gradients, which significantly improves the ability to learn complex policies in high-dimensional state spaces.

The design of the DDPG algorithm is based on the Actor-Critic framework[15], where the Actor network is responsible for determining the optimal actions to take in a given state, while the Critic network evaluates the value of these actions. Unlike traditional Actor-Critic methods, the DDPG algorithm utilizes a deep neural network to approximate the Actor's strategy and the Critic's value function, which allows the algorithm to handle high-dimensional inputs that are difficult to deal with in traditional methods[16]. The core idea of the DDPG algorithm, shown in Figure 2, is the simultaneous use of two neural networks: an Actor network, which determines the action to be taken in a given state, and a Critic network, which evaluates the value of the state-action pairs under the current policy. The Actor network outputs a deterministic policy, which is the selection of an optimal action in each state, while the Critic network evaluates the expected payoff of that action[17]. The DDPG algorithm provides an effective solution to the problem of reinforcement learning in continuous action space by combining deep learning and policy gradient methods. With the introduction of goal networks, experience playback and noise exploration strategies, the DDPG algorithm demonstrates excellent performance in dealing with high-dimensional state spaces and complex policy learning[18]. As the field of deep reinforcement learning continues to evolve, DDPG plays an important role in areas such as automation control, robotics and game intelligence[19,20].

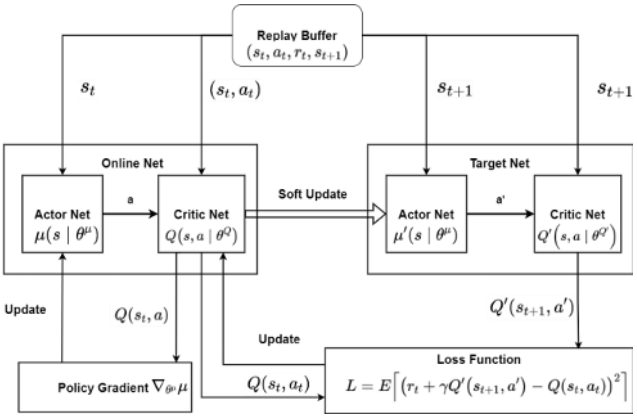


Figure 2: DDPG Algorithm Architecture

III. System Model

A. *DAG Model*

In DAG task scenarios, efficient scheduling of DAG tasks is essential to ensure high performance execution. As shown in the DAG task structure, each subtask is represented by an independent node, and the set of tasks is represented by $V = \{v_1, v_2, ..., v_n\}$, and the interdependencies between subtasks are represented by directed edges between nodes $E = \{e_1, e_2, ..., e_i\}$, and since dependency data needs to be passed between each task node with dependency relationship, the size of the dependent data between subtasks is represented by the weights of these directed edges.

In the DAG task model, tasks that do not have any antecedent task nodes are defined as entry task nodes, while those that do not have successor task nodes are defined as exit task nodes. To simplify the complexity of the model, this study assumes that each DAG task process contains only one entry task node as well as one exit task node. If more than one entry or exit task node exists, a virtual pseudo-entry node or pseudo-exit node is introduced into the model, and the computational complexity of these pseudo-nodes and the amount of data dependency on other subtasks are considered to be zero.

In the simplified example of the DAG cloud task shown in Figure 3, except for the entry task node identified as “Start” and the exit task node identified as “Exit”, all other task nodes have at least one parent or one child node, and some

task nodes may have at least one parent or one child node, and some task nodes may have at least one parent or one child node. All other task nodes, except the "Start" entry task node and the "Exit" exit task node, have at least one parent or one child, and some task nodes may have more than one parent and more than one child. This dependency relationship between tasks creates inherent priority constraints that ensure sequential and correct scheduling, i.e., any child task can only start execution after all its parents have successfully completed and passed all dependent data.

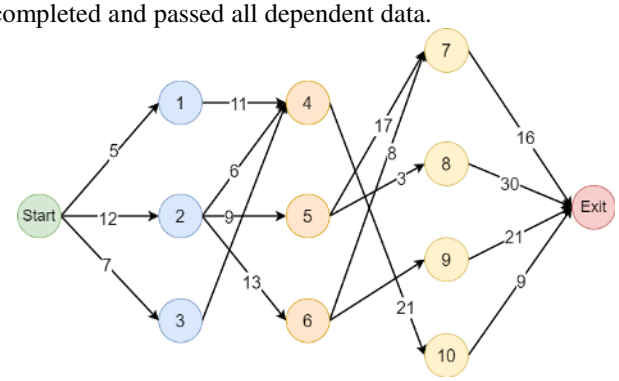


Figure 3: DAG Task

B. Task scheduling Model

As shown in Figure 4, the cloud computing task scheduling model for DAG tasks contains three key components: user-defined DAG tasks, scheduling center, and cloud data center. In the task scheduling system model, the user is responsible for defining and submitting the DAG task and its associated resource requirements; the scheduling center evaluates the priority of the DAG task and completes the resource allocation based on the requirements, and it is responsible for deciding how to assign the task to the available resources in the most efficient way; and the cloud data center, which consists of a series of heterogeneous VM resources, is responsible for executing the task and returning the results to the scheduling center.

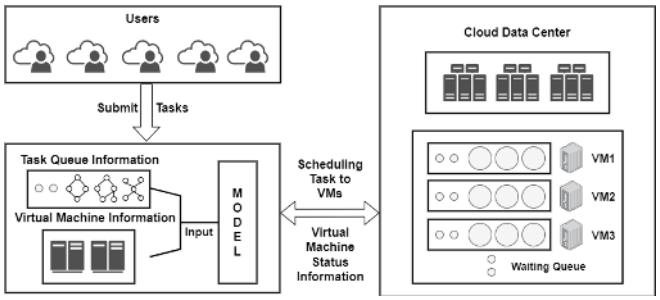


Figure 4: Task scheduling Model

A DAG task contains multiple subtasks, assuming that a DAG task contains n subtasks, which can be expressed as Equation 3.1. The attribute composition of subtasks mainly contains the number of instructions, the data size, the computational resource requirement, and the task priority. These attributes are indispensable considerations in the task scheduling process, according to which the scheduler needs to prioritize tasks and assign them to the most suitable server resource pools to optimize the system performance and resource utilization. the DAG task information is represented as shown in Equation 3.2:

$$job_i = \{task_{i,1}, task_{i,2}, task_{i,3}, \ldots, task_{i,n}\} \quad (3.1)$$

$$task_{i,j} = \{mi_{size}, data_{size}, cpu_{demand}, mem_{demand}, level_p\} \quad (3.2)$$

The server composition of a cloud data center is defined by instruction processing capability, CPU and memory resources, network transmission speed, and operational power. The parameterization used by the server to support the DAG task is represented as in Equation.3.3.

$$server_i = \{mips_{speed}, cpu_{capacity}, ram_{capacity}, net_{speed}, power_{work}\} \quad (3.3)$$

A cloud data center can be described as a set of n heterogeneous servers serving prioritized tasks, each configured with different processing capabilities, resource ratios, and operating power.

C. Optimization Model

The algorithmic model optimization objective consists of three parts, average DAG task completion time, standard deviation of server resource utilization and server operation power consumption.

1. Average completion time

Average time to completion (AFT) is one of the key metrics to measure the performance of task scheduling algorithms and is particularly important in cloud computing resource scheduling. It refers to the average time for all subtasks to complete their execution. Specifically, it is the sum of the completion times of all subtasks divided by the number of subtasks. For a DAG job, the predecessor subtasks with which it is bound should have been executed before the subtasks are executed, and the earliest idle time of the subtasks on the server is shown in Equation 3.4. The runtime of a task is the ratio of the number of instructions executed by the task to the speed at which the server executes the instructions, and the data transfer time is the ratio of the amount of data for the task to the network bandwidth of the server. The completion time of the predecessor tasks of the task is shown in Equation 3.5, where pred indicates all the predecessor tasks of the task, and finish time indicates the completion time of the sub-task v. If there is no predecessor task for the task, it is zero. Based on the above analysis, as shown in Equation 3.6, the actual start time can be derived as the earliest idle time and the completion time of the predecessor tasks plus the transmission time of the sub-task v The start time of subtask v is the ratio of the number of task execution instructions to the server execution instruction speed. The actual start time is the maximum value of the earliest idle time and the completion time of the predecessor task plus the transmission time in the server, and the actual completion time of the task is the actual start time of the task plus the actual execution time, so as to obtain the actual completion time of the subtasks in the server, as shown in Equation 3.7:

$$task_i^{\text{runTime}} = \frac{task_{i,j}^{mi}}{server_i^{mipSpeed}}, \quad task_{i,j}^{\text{transTime}} = \frac{task_{i,j}^{data}}{server_i^{\text{netSpeed}}} \quad (3.4)$$

$$predFT_{i,j} = \max_{v_k \in pred(task_{i,j})} finishTime(v_k) \quad (3.5)$$

$$startTime_{i,j}^{k} = max\left(server_{k}^{freeTime}, predFT_{i,j} + task_{i,j}^{transTime}\right) \quad (3.6)$$

$$finishTime_{i,j}^{k} = startTime_{i,j}^{k} + task_{i,j}^{runTime} \quad (3.7)$$

For a single job, the goal is to minimize the maximum completion time of all subtasks at server {server}_k. Based on the above conditions, the maximum completion time of task job can be obtained as the completion time of export task n. The finishTime is denoted as the completion time of the nth subtask of the i task. the export task, at server k, as shown in Equation 3.8:

$$maxFinishTime_{i} = finishTime_{i,n}^{k} \quad (3.8)$$

For the whole cloud computing platform, the goal is to make the waiting time of all the tasks as short as possible, so it is necessary to find a program that can make the average completion time of all the tasks is the shortest, assuming that there are n tasks, the average completion time of the task is denoted as AFT, and the calculation is shown in Equation 3.9:

$$AFT = \frac{1}{n}\sum_{i=1}^{N} maxFinishTime_{i} \quad (3.9)$$

2. Standard deviation of resource utilization rates

In server clusters, resource load balancing is critical for improving performance, reducing energy consumption and optimizing resource utilization. In this paper, we use Standard Deviation of Resource Utilization (SDRU) as a metric, which measures the degree of dispersion in the distribution of server resource utilization, thus reflecting the overall load balancing. Assuming that there are n servers, and the corresponding resource utilization of each server, the average value is calculated based on the cpu resource utilization and memory utilization, and the formula is as follows Equation 3.10:

$$\bar{x}_{cpu} = \frac{1}{n}\sum_{i=1}^{n} server_{i}^{cpu}, \bar{x}_{ram} = \frac{1}{n}\sum_{i=1}^{n} server_{i}^{ram} \quad (3.10)$$

The resource utilization standard deviation calculates the difference between each server's resource utilization and the average, calculated as follows Equation 3.11:

$$SDRU = \sqrt{\frac{1}{n}\left[\sum_{i=1}^{n}\left(server_{i}^{cpu} - \overline{x_{cpu}}\right)^{2} + \sum_{i=1}^{n}\left(server_{i}^{ram} - \overline{x_{ram}}\right)^{2}\right]} \quad (3.11)$$

3. Server operating power consumption

As far as the cloud platform power consumption is concerned, since the power of the servers in a cloud computing system varies, the operational power consumption generated by a single server is the product of the power and the time spent executing tasks on server. Assuming that there are m servers and n tasks are executed on the kth server, Equation 3.12 can be expressed as the sum of the ratio of the number of instructions executed for all the tasks to the execution speed of the server as shown in Equation 3.13. The power consumption of the entire cluster is the sum of the product of the power consumption of all servers and the server execution time:

$$taskRunTime^{k} = \sum_{i=1}^{n} \frac{task_{i}^{mi}}{server_{k}^{miSpeed}} \quad (3.12)$$

$$POWER = \sum_{k=1}^{m} taskRunTime^{k} * server_{k}^{power} \quad (3.13)$$

D. Algorithm Model

Markov Decision Process (MDP) was introduced earlier as a mathematical model to describe systems with stochasticity and decision making. The Markov Decision Process consists of several basic constituent elements: state space, action space, and reward function. In this section, the MDP model is developed based on the task scheduling system model as follows:

1. State space design

For the state of job, for any moment t the system is able to obtain information about all the ready subtasks in the set of submitted tasks, and each subtask information can be represented as task, where i represents the i task in the set of submitted tasks and j represents the j subtask of task i. The state of each subtask consists of the number of task instructions, data size, occupied cpu resources, occupied memory resources, network bandwidth speed and task priority. The state of each subtask consists of the number of task instructions, data size, occupied cpu resources, occupied memory resources, network bandwidth speed and task priority, and the state of each server consists of the speed of the server executing instructions, the load of cpu resources, the load of memory resources, the speed of the network bandwidth and the operating power, and the state space is shown in Equation 3.14:

$$state_{i} = \begin{bmatrix} task_{i,1}^{miSize}, task_{i,1}^{dataSize}, task_{i,1}^{pLevel}, task_{i,1}^{cpuDemand}, \\ task_{i,1}^{ramDemand}, ..., task_{i,n}^{miSize}, task_{i,n}^{dataSize}, task_{i,n}^{pLevel}, \\ task_{i,n}^{cpu}, task_{i,n}^{ram}, ..., server_{i}^{mipSpeed}, server_{i}^{netSpeed} \\ server_{i}^{cpuCapacity}, server_{i}^{ramCapacity}, server_{i}^{power}, ... \end{bmatrix} \quad (3.14)$$

2. Action space design

Assuming that there are J servers, the action space is composed of the set of numbers of these J servers. Therefore, an action can be represented as assigning subtask j of task i to server number k. The action space is represented by Equation 3.15:

$$action_{i,j,k} = \left[server_{1}, server_{2}, \ldots, server_{k}\right] \quad (3.15)$$

3. Reward function design

According to the objective optimization function proposed by Algorithm Model, the reward function consists of the average task completion time AFT, the standard deviation of the server's resource utilization SDRU, and the power consumption of the cluster operation POWER, and the reward function is as shown in Equation 3.16 by setting up the different weighting system in order to optimize the different indexes.

$$\text{reward} = \alpha * AFT + \beta * SDRU + \gamma * POWER \quad (3.16)$$

IV. METHOD

The efficiency of task scheduling is crucial in cloud computing environments targeting DAG tasks. The Deep Deterministic Policy Gradient (DDPG) based algorithm proposed in this paper aims to allocate optimal server

resources for DAG tasks through a deep reinforcement learning approach. The following is the basic flow of task scheduling based on the DDPG algorithm:

1. Initialization environment: define the simulation environment, initialization includes DAG task parameters, heterogeneous server parameters, etc. Before the simulation starts, set the DDPG algorithm hyper-parameters, including the learning rate, discount factor, noise parameter, soft update factor, as well as the size of the empirical playback buffer for the learning process and the update frequency of the target network.

2. defining task queue and initial state: based on the task scheduling system model detailed in Subsection 4.1.2, a task queue is created to receive user-submitted DAG tasks, and the initial state of the cloud computing environment is obtained, including the characteristics of the tasks and the current load of the server.

3. environment interaction: generating actions based on Actor online policy network, and facilitating intelligences to explore unknown environments based on action exploration strategy. Execute selected actions in the task scheduling system, assign tasks to VMs and observe the response of the environment, including the reward obtained and the next state.

4. storing experience: storing experience tuples (current state, action, reward, next state) into the experience playback buffer. Sampling from experience playback buffer: Randomly sample a batch of experience data from the experience playback buffer for network training.

5. Updating the network: the Critic online Q network utilizes the actual rewards and the target Q network outputs to compute the loss and update its parameters, while the Actor online policy network adjusts the action selection by updating its policy gradient based on the value assessment of the Critic online Q network. In order to maintain the stable learning of the algorithm, the parameters of the target network are periodically modified for the online network through a soft update mechanism, which maintains the smoothness of the learning process.

In order to improve training stability, this paper proposes supervised training methods to train Actor networks offline. By filtering the training data whose reward is less than the average reward of a batch of samples, they reflect the potential for optimizing the performance of the system in the current state. We add a new loss function to train the Actor network, the loss function is shown in Equation 4.1:

$$Loss = \frac{1}{n}\sum_{i}\left(w(r_i)\left(a_i - \mu\left(s_i \mid \theta^{\mu}\right)\right)^2 - \lambda Q\left(s_i, a_i \mid \theta^{Q}\right)\right) \quad (4.1)$$

In this loss function, $w(r_i)$ is a weighting function based on the reward r, whose purpose is to adjust the impact of each sample on the loss based on the size of the reward. Whereas λ is the balance coefficient, $Q\left(s, a_i \mid \theta^{Q}\right)$ is the Q value predicted by the Critic network for taking action a_i in state s_i, which is used to regulate the action prediction error and the expected reward estimated by the Actor network. By adding this new loss function to the Actor training process, it not only reduces the prediction error, but also guides the Actor network to update in the direction of actions that are likely to receive higher future rewards, thus providing an effective strategy to improve the training process and stability of the Actor network. By integrating such a strategy into a task scheduling algorithm for cloud computing, we are able to parametrically show the application of DDPG in real complex systems. The detailed design of the cloud computing task scheduling algorithm based on DDPG is shown in Algorithm 1:

Algorithm 1: DDPG-Based Task Scheduling Algorithm

1: Initialize online Critic $Q\left(s,a \mid \theta^{Q}\right)$ and Actor networks $\mu\left(s,a \mid \theta^{\mu}\right)$ with weights θ^{Q} and θ^{μ}.
2: Initialize the Critic and Actor networks in the target network $\mu' : \theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
3: Initializing task scheduling system tasks and server parameters
4: Hyperparameters such as experience playback pool $\Re$, soft update factor ρ, reward discount γ, randomness noise in initialization $Noise$
5: **for** each episode **do**:
6: get initialization state s_1 from the environment
7: **for** each episode **do**:
8: get the task state and current server load from the environment
9: pick an action by adding $a = \mu\left(s \mid \theta^{\mu}\right) + noise_t$ noise to the actor network
10: execute at in the task scheduling system and observe the reward value rt and the environment
11: stored in the experience pool (s_t, a_t, r_t, s_{t+1})
12: set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \theta^{Q'}\right)$
13: update the Critic network based on critic loss: $L = \frac{1}{N}\sum_{i}\left(y_i - Q\left(s, a \mid \theta^{Q}\right)\right)^2$
14: select r less than the mean from N data n
15: add a loss function to update the Actor network: $L = \frac{1}{n}\sum_{i}(w(r_i)(a_i - \mu(s_i \mid \theta^{\mu}))^2 - \lambda Q(s_i, a_i \mid \theta^{Q}))$
16: update the Actor network based on the gradient of the actor: $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla Q\left(s,a \mid \theta^{Q}\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta}\mu\left(s \mid \theta^{\mu}\right)\Big|_{s_i}$
17: update the target network: $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
18: **end**
19: **end**

V. EXPERIMENTS

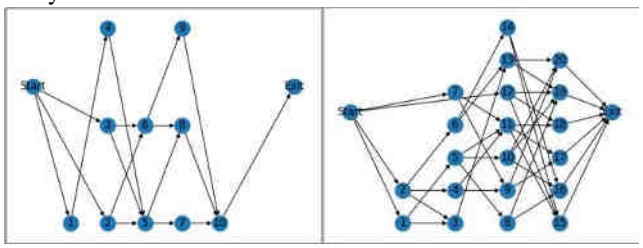
A. Datasets

In this paper, a method for randomly generating directed acyclic graphs (DAGs) is proposed for modeling workflow task scheduling of different sizes and complexities. With this approach, DAGs with various characteristics can be flexibly created, and thus task scheduling strategies can be evaluated and optimized.

The number of randomly selected DAG nodes is set to

range from 10 to 30, and the maximum out-degree of the nodes varies from 1 to 5. Two parameters are also defined to control the shape and regularity of the DAG, where \alpha determines the depth of the DAG and \beta affects the standard deviation of the distribution of the nodes at each layer.

Finally, in order to evaluate different task scheduling algorithms, this paper generates a large number of training and test datasets using the methodology described above. Separate training and test sets were created, where each set contained 1000 DAGs of different sizes. the structure, duration and resource requirements of these DAGs were saved in NumPy array format for subsequent processing and analysis.



(a) DAG of 10 subtask (b) DAG of 20 subtask

Figure 5: DAG Generation Tasks

B. *Parameters*

The neural network structure of the DDPG deep reinforcement learning method proposed in this paper adopts a 5-layer network structure, using Relu as the activation function between the hidden layers and Sigmoid as the activation function in the output layer, and the other parameters in the network are shown in Table 1:

Table 1: Parameterization of the DDPG algorithm

Parameter	Value	Meaning
lr_actor	0.0005	Actor learning rate
lr_critic	0.001	Critic learning rate
update_timestep	1000	Network update step
tau	0.01	Soft update frequency
batch_size	128	Batch size
epsilon	0.95	Exploration rate
gamma	0.99	Exploration rate decay
epsilon_min	0.1	Minimum exploration rate
step	10000	Training step

C. *Experiment Result*

As shown in Figure. 6, the horizontal coordinate represents the number of subtasks in the DAG task and the vertical coordinate represents the average completion time of the DAG task. The average task completion time of our proposed supervised training STDDPG algorithm significantly outperforms other scheduling algorithms. The traditional DDPG algorithm performs second only to the STDDPG algorithm in the experiments. Although the SJF algorithm is able to reduce the waiting time of short jobs in some cases, it does not perform well in the experiments. The Random algorithm performs task scheduling based on random choices and does not take into account the priority relationship of the tasks, and this randomness leads to the lower scheduling efficiency of the Random algorithm in the experiments and the longer average task completion time. The RR scheduling algorithm does not have a significant effect due to the use of polling scheduling . In 10DAGs scenario STDDPG algorithm reduces the average task completion time by 5.3% compared to DDPG algorithm, 18.3% compared to Random algorithm, 6.5% compared to SJF algorithm and 16.4% compared to RR algorithm.

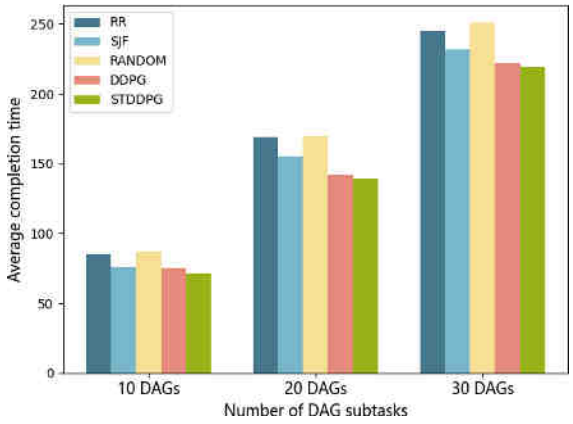


Figure 6: Average completion time Analysis

As shown in Figure. 7, the horizontal coordinate indicates the number of subtasks in the DAG task and the vertical coordinate indicates the standard deviation of server resource utilization. The experimental results indicate that the DDPG algorithm performs second only to the STDDPG algorithm in terms of standard deviation of resource utilization. The RR algorithm performs the worst in terms of standard deviation of resource utilization. The RR algorithm performs task scheduling through the time-slice rotation strategy, which ensures fairness in task scheduling but results in certain tasks not being completed in time due to long waiting time, thus affecting the overall utilization of resources. The STDDPG algorithm proposed in this study reduces the standard deviation of resource utilization by 7.4% compared to the DDPG algorithm, 28.5% compared to the RR algorithm, 19.3% compared to the Random algorithm, and 10.7% compared to the SJF algorithm for the scenario of 10DAGs, and the combined analysis , the STDDPG scheduling algorithm proposed in this study effectively improves the resource utilization in D-task scenarios.

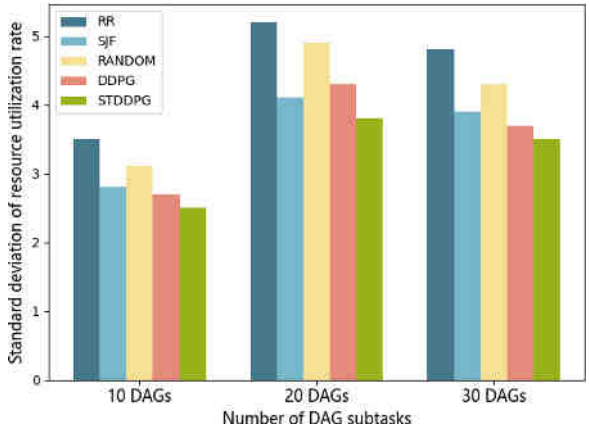


Figure 7 Standard deviation of resource utilization rate Analysis

This experiment provides an in-depth evaluation of the

application of the STDDPG algorithm to a multi-objective optimization problem. The core objective is to achieve high efficiency in task scheduling and fairness in resource utilization, i.e., to minimize the average task completion time and to optimize the standard deviation of server resource utilization. In order to achieve these two objectives in a balanced way, this study designs and implements an integrated reward function, which takes the weighted sum of the average response time of the task and the standard deviation of the resource utilization as the immediate reward of the intelligences, where the weight of both objectives is set to 0.5. The experimental environment is constructed based on the DAG (Directed Acyclic Graph) task model, in which each task consists of 10 sub-tasks, and all the tasks are computational center. The experimental results are shown in 8. The STDDPG algorithm effectively controls the standard deviation of resource utilization while minimizing the average task completion time. Compared with the single-objective optimization algorithm, the STDDPG algorithm generally demonstrates significant advantages over the traditional scheduling algorithm in the overall experimental results. Compared to the DD algorithm where the average task response time is reduced by 9.3% and the standard deviation of resource utilization is reduced by 7.3%, the DDPG algorithm shows a significant improvement in both the reduction of average response time and the reduction of standard deviation of resource utilization.

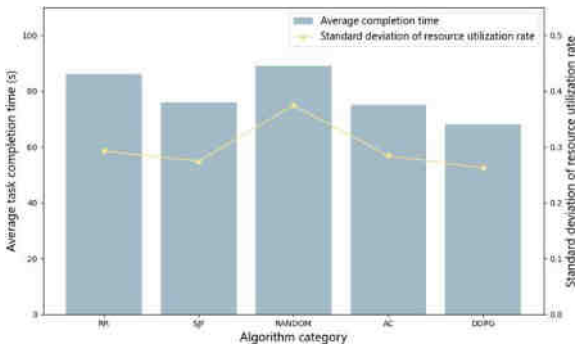


Figure 8 Standard deviation of resource utilization rate Analysis

VI. CONCLUSION

In this paper, we present a novel supervised deep reinforcement learning algorithm, dubbed Supervised Training Deep Deterministic Policy Gradient (STDDPG). We have constructed a task model based on Directed Acyclic Graphs (DAG) and a scheduling system model, for which we have formulated an objective optimization function. The paper then redefines the state space, action space, and rewards function, and enhances the agent's ability to explore the environment through improved action exploration strategies. Incorporating supervised training refines the Actor network's loss function within a DDPG-based scheduling algorithm, bolstering the network's learning capabilities in complex and dynamic environments and increasing the algorithm's flexibility and accuracy. The refined algorithm achieves a balanced optimization of objectives, efficiently harmonizing task average completion times with server resource utilization while maintaining robustness.

This research delves deeply into the theoretical analysis and modeling of cloud computing task scheduling and empirically validates the effectiveness of the proposed algorithm. The innovative outcomes discussed in this article offer a fresh perspective and method for resolving task scheduling challenges in cloud computing settings, holding significant theoretical and practical significance for the advancement and application of cloud computing technologies.

REFERENCES

[1] Dong T, Xue F, Xiao C, et al. Workflow scheduling based on deep reinforcement learning in the cloud environment[J]. Journal of Ambient Intelligence and Humanized Computing, 2021, 12(12): 10823-10835.
[2] Wei Y, Pan L, Liu S, et al. DRL-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds[J]. IEEE Access, 2018, 6: 55112-55125.
[3] Siddesha K, Jayaramaiah G V, Singh C. A novel deep reinforcement learning scheme for task scheduling in cloud computing[J]. Cluster computing, 2022, 25(6): 4171-4188.
[4] Wang T, Liu Z, Chen Y, et al. Load balancing task scheduling based on genetic algorithm in cloud computing[C]//2014 IEEE 12th international conference on dependable, autonomic and secure computing. IEEE, 2014: 146-152.
[5] Awad A I, El-Hefnawy N A, Abdel_kader H M. Enhanced particle swarm optimization for task scheduling in cloud computing environments[J]. Procedia Computer Science, 2015, 65: 920-929.
[6] Li K, Xu G, Zhao G, et al. Cloud task scheduling based on load balancing ant colony optimization[C]//2011 sixth annual ChinaGrid conference. IEEE, 2011: 3-9.
[7] Gan G, Huang T, Gao S. Genetic simulated annealing algorithm for task scheduling based on cloud computing environment[C]//2010 International Conference on Intelligent Computing and Integrated Systems. IEEE, 2010: 60-63.
[8] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).
[9] Cheng M, Li J, Nazarian S. DRL-Cloud: Deep Reinforcement Learning-Based Resource Provisioning and Task Scheduling for Cloud Service Providers[J].
[10] François-Lavet V, Henderson P, Islam R, et al. An introduction to deep reinforcement learning[J]. Foundations and Trends® in Machine Learning, 2018, 11(3-4): 219-354.François-Lavet V, Henderson P, Islam R, et al. An introduction to deep reinforcement learning[J]. Foundations and Trends® in Machine Learning, 2018, 11(3-4): 219-354.
[11] Rittinghouse J W, Ransome J F. Cloud computing: implementation, management, and security[M]. CRC press, 2017.
[12] Attaran M, Woods J. Cloud computing technology: improving small business performance using the Internet[J]. Journal of Small Business & Entrepreneurship, 2019, 31(6): 495-519.
[13] Srivastava P, Khan R. A review paper on cloud computing[J]. International Journal of Advanced Research in Computer Science and Software Engineering, 2018, 8(6): 17-20.
[14] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//International conference on machine learning. Pmlr, 2014: 387-395.
[15] Duan J, Guan Y, Li S E, et al. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors[J]. IEEE transactions on neural networks and learning systems, 2021, 33(11): 6584-6598.
[16] Ru J, Keung J. An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems[C]//2013 22nd Australian Software Engineering Conference. IEEE, 2013: 78-87.
[17] Zhou G, Tian W, Buyya R. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions[J]. arXiv preprint arXiv:2105.04086, 2021.
[18] Peng Z, Lin J, Cui D, et al. A multi-objective trade-off framework for cloud resource scheduling based on the deep Q-network algorithm[J]. Cluster Computing, 2020, 23: 2753-2767.
[19] Cheng Z, Min M, Liwang M, et al. Multiagent DDPG-based joint task partitioning and power control in fog computing networks[J]. IEEE Internet of Things Journal, 2021, 9(1): 104-116.
[20] Zheng T, Wan J, Zhang J, et al. Deep reinforcement learning-based workload scheduling for edge computing[J]. Journal of Cloud Computing, 2022, 11(1): 3.