International Journal of Computing and Digital Systems
2025, VOL. 17, NO. 1, 1–11

http://dx.doi.org/10.12785/ijcds/1571017581

Efficient Task Scheduling in Cloud Using Double Deep Q-Network

S Radhika[1,3], Sangram Keshari Swain[1], S Adinarayana[2] and BSSV Ramesh Babu[4]

[1]*Department of CSE, Centurion University of Technology and Management, Odisha, India*
[2]*Department of CSSE, Andhra University college of Engineering, Visakhapatnam, Andhra Pradesh, India*
[3]*Department of CSE, Anil Neerukonada Institute of Technology, Visakhapatnam, Andhra Pradesh, India*
[4] *Department of ECE, Raghu Engineering college, Visakhapatnam, Andhra Pradesh, India*

Received 9 April 2024, Revised 28 November 2024, Accepted 5 December 2024

Abstract: Cloud computing has transformed data management with its scale and flexibility. Cloud resources are transient and diversified, making task scheduling difficult. This paper proposes Double Deep Q-Network (DDQN) reinforcement learning model to solve the cloud computing task scheduling problem. Double Deep Q-Network (DDQN) is a powerful reinforcement learning system that improves on Deep Q-Networks (DQN). The target network and the online network are the two distinct neural networks that DDQN presents. To create a more consistent and less unpredictable learning process, the target network is updated on a regular basis to imitate the Q-value estimations of the online network. Traditional DQN can have problems with overestimation bias, which is something that this dual-network architecture helps to alleviate. DDQN is a reliable and efficient tool for solving complex reinforcement learning problems. It excels in learning optimal strategies through iteratively improving its Q-value estimations.

Keywords: Cloud computing, Data management, Task scheduling, Double Deep Q-Network (DDQN), Reinforcement learning, Deep Q-Networks (DQN), Target network, Online network

1. Introduction

Task scheduling is a crucial element in the dynamic field of cloud computing, playing a major role in ensuring the efficient operation of services hosted on the cloud [1]. The core of task scheduling is its capacity to strategically distribute computing work among the available cloud resources. This process directly affects the performance, cost efficiency, and user happiness in cloud settings [2]. Conventional scheduling techniques, primarily static or heuristic-driven, have historically been the primary approach to addressing this problem. Nevertheless, as cloud computing environments become more intricate and ever-changing, these traditional methods frequently prove inadequate. The need for a scheduling solution that is more flexible and intelligent is highlighted by the inability to adjust to changing workloads and resource availability in real-time, as well as the issues provided by the unpredictable nature of cloud needs. This context provides the opportunity to investigate sophisticated machine learning methods, including those related to Deep Reinforcement Learning (DRL), as possible catalysts for transforming task scheduling in cloud systems [3].

The incorporation of Deep Reinforcement Learning (DRL) into cloud task scheduling offers the potential for algorithms that possess the ability to acquire knowledge, adjust, and make choices in ever-changing and unpredictable settings [4]. Double Deep Q-Network (DDQN) is a notable method among the different DRL techniques, mostly because of its improved stability and efficiency in learning policies. This is particularly advantageous in situations that involve intricate state-action spaces. The strength of DDQN lies in its capacity to separate the selection and evaluation of activities, which represents a notable advancement compared to its predecessor, the Deep Q-Network (DQN) [5]. This characteristic renders it very appropriate for the complexities of cloud task scheduling, wherein decisions must be made taking into account a multiplicity of criteria such as job priority, resource availability, execution time, and cost. Within a cloud environment characterized by unpredictable fluctuations in resource demands and varying availability of computing resources, a DDQN-based approach can adaptively allocate tasks to optimize multiple objectives, such as minimizing delays, balancing workloads, and maximizing resource utilization [6].

E-mail address: radhikabssv@gmail.com, sangrambapun@gmail.com, prof.sadinarayana@andhrauniversity.edu.in, rameshbssv@gmail.com

The investigation of DDQN in cloud task scheduling not only expands the possibilities in cloud computing research but also offers concrete enhancements in the operational efficiency of cloud services, potentially resulting in more economical and user-focused cloud computing models.

The main focus of this study is to provide a task scheduling mechanism that is efficient and can adjust to the ever-changing and uncertain conditions of cloud settings. The goal is to minimize the time it takes to complete tasks and maximize the use of resources, while also maintaining fairness and preventing conflicts over resources. The suggested approach utilizes Double Deep Q-Network (DDQN) [7] to efficiently manage job scheduling in cloud environments. The capability of DDQN to manage input spaces with a large number of dimensions and acquire optimal policies in intricate situations makes it highly suitable for cloud job scheduling [8]. The suggested Double Deep Q-Network (DDQN) model is specifically developed to iteratively acquire knowledge and adjust its scheduling approach in response to real-time input obtained from the cloud environment.

This work presents the following significant contributions:

- The framework presented is a novel approach to job scheduling in cloud computing, utilizing a DDQN-based method.
- The text provides a thorough examination of the effectiveness of the suggested framework in comparison to conventional scheduling methods.
- The study offers valuable information on the scalability and adaptability of DDQN in managing various cloud computing situations.

The subsequent sections of the paper are structured in the following manner: Section II provides an overview of previous research conducted in the field of cloud task scheduling and Deep Reinforcement Learning (DRL). Section III provides a comprehensive explanation of the methodology employed in the proposed task scheduling system, which is based on the Double Deep Q-Network (DDQN) approach. Section IV outlines the experimental configuration and assessment criteria. Section V presents the findings and evaluates the effectiveness of the proposed system in comparison to established approaches. Section VI serves as the final section of the work, providing a conclusion and delineating potential avenues for further research.

2. Literature Survey

Kaixuan Kang al [9] presented an energy-efficient cloud computing job scheduling system called Adaptive Deep Reinforcement Learning-based (ADRL). Initially, we provide a Change Detection method designed to identify significant changes in the workload. Using this foundation, we developed an Automatic Generation network that can adaptively modify the discount factor of Deep Reinforcement Learning (DRL) based on variations in workload. This allows for quicker and more precise learning. We are now using adaptive Deep Reinforcement Learning (DRL) to determine the most effective strategy for dispatching incoming user requests. The objective is to decrease task response time and increase resource usage by optimizing the incentive system.

Mohan Sharma et al [10] presented a novel approach for optimizing task scheduling in an energy-efficient manner by using supervised neural networks. The primary objective is to minimize the makespan, energy consumption, execution overhead, and the number of active racks. The proposed artificial neural network-based scheduler utilizes incoming tasks and the existing state of the cloud environment as input to anticipate the most suitable computing resource for a particular job as output, aligning with our objective. Shashank Swarup et al [11] addressed the issue of job scheduling for cloud-based applications and seeks to reduce the computational expenses while adhering to resource and schedule limitations. To achieve this objective, we suggest using a clipped double deep Q-learning algorithm that incorporates the target network and experience relay approaches, in addition to leveraging the reinforcement learning strategy.

Zhou Zhou et al [12] suggested a new method called MGGS, which is a combination of a modified genetic algorithm (GA) and a greedy approach. The approach described utilizes a modified genetic algorithm (GA) in conjunction with a greedy strategy to enhance the optimization of the job scheduling process. Contrary to current algorithms, MGGS has the ability to discover an optimum solution with a reduced amount of repetitions. In order to assess the effectiveness of MGGS, we conducted a comparative analysis of its performance against several established algorithms. This evaluation was based on metrics such as the overall completion time, average response time, and quality of service (QoS) characteristics.

Huanhuan Hou et al [13] conducted a study and analysis of energy consumption models used for scheduling objectives, provide a summary of the Deep Reinforcement Learning (DRL) algorithms utilized in existing research, and quantitatively assess the variations in Markov Decision Process components. In addition, we provide a concise overview of the experimental platforms, datasets, and neural network architectures used in the DRL method.

Tingting Dong et al [14] proposed a technique, RLTS, utilizes a deep reinforcement learning architecture to dynamically schedule tasks with priority connection to cloud servers. Its objective is to reduce the time used for task execution. Meanwhile, the Deep-Q-Network, a kind of deep reinforcement learning method, is used to address the issue of complexity and high dimensionality. Zihui Zhao et al [15] presented an intelligent cloud task scheduler based on deep reinforcement learning (DRL). This scheduler

generates optimum scheduling decisions solely based on learning from its own experience, without relying on any past information. The task scheduling issue is modeled as a dynamic optimization problem with constraints. To identify the best task assignment solution that satisfies performance and cost limitations, we use the deep deterministic policy gradients (DDPG) network. We introduce a correlation-aware state representation approach to capture the fundamental attributes of requests, and we construct a dual reward model to discover the ideal strategy for task allocation.

Yaoyao Ping et al [16] presented two different sequence creation techniques for the purpose of constructing scheduling sequences of many composite jobs before scheduling. Additionally, a Deep Q-Networks (DQN)-based scheduling method and a Double DQN-based scheduling algorithm are presented, respectively, in conjunction with this. Saravanan Muniswamy et al [17] presented a hybrid optimal and deep learning strategy for dynamic scalable task scheduling (DSTS) in a container cloud environment. In order to increase the virtual resources of containers, we propose an enhanced version of the multi-swarm coyote optimization (MMCO) technique, which enhances the fulfillment of client service level agreements. In order to ensure scheduling based on priority, we develop a modified pigeon-inspired optimization (MPIO) technique for grouping tasks together and a rapid adaptive feedback recurrent neural network (FARNN) for allocating pre-virtual CPUs. The task load monitoring system is constructed using a deep convolutional neural network (DCNN), enabling the implementation of dynamic priority-based scheduling.

Qirui Li et al [18] suggested a cloud-based Time-Space Resource Allocation (TSRA) framework using deep learning (DL) techniques. The solution addresses the challenges of managing various task queues and virtual machine (VM) clusters in TSRA issues by integrating multiple deep neural networks (DNNs) as the job scheduler for the cloud system. The DNNs are partitioned into an exploration component and an exploitation component. During each scheduling time step, the model stores the most optimal outputs of all scheduling rules from each DNN in the experienced sample memory pool (SMP). Periodically, random training samples are chosen from the SMP to train each DNN in the exploitation section.

Traditional scheduling algorithms often lack the ability to dynamically adapt to real-time changes in cloud environments. Many existing solutions struggle to scale effectively in large cloud environments with diverse and complex workloads. Deep Q-Networks (DQN) tend to suffer from overestimation bias, leading to suboptimal policy learning. Conventional heuristic-driven and static scheduling techniques often fail to optimize resource utilization effectively.

The proposed DDQN model incorporates real-time feedback and continuous learning, enabling it to adapt to fluctuating workloads and resource availability dynamically. The DDQN model's architecture, leveraging neural networks and reinforcement learning, is designed to handle large state-action spaces, ensuring scalability in extensive cloud infrastructures. The use of Double Deep Q-Network (DDQN) mitigates this issue by separating action selection and evaluation, resulting in more accurate Q-value estimations and improved policy performance. The DDQN model optimizes multiple objectives such as minimizing delays, balancing workloads, and maximizing resource utilization, leading to more efficient use of cloud resources.

3. Proposed Model: Double Deep Q-Network (DDQN)

Deep Q-Network (DQN) is a reinforcement learning algorithm. The DQN algorithm revolutionized the field by integrating traditional Q-learning with deep neural networks. This breakthrough allowed for the effective management of complex state spaces, such as those found in video games. Nevertheless, the Deep Q-Network (DQN) exhibited a proclivity for overestimating Q-values as a result of the intrinsic noise present in the training data, hence resulting in inefficient policy learning.

A. Neural Network Architectures

Neural networks are now a crucial tool for maximizing cloud resources because they can effectively handle intricate and ever-changing settings, enabling them to make intelligent judgments based on data. Neural networks are highly proficient in detecting patterns in extensive datasets, making them essential in cloud environments that are characterized by immense volumes of data. By examining historical usage patterns, performance indicators, and workload characteristics, they are able to predict future needs and determine the necessary resources. The ability to forecast future outcomes allows for proactive allocation of resources, resulting in reduced waste and improved efficiency. Cloud infrastructures are inherently volatile, with changing demands and diverse workload characteristics. Neural networks, particularly deep learning models, have the ability to adapt to these changes by constantly acquiring new knowledge and modifying their forecasts and choices. They have the ability to efficiently allocate compute power, storage, and network bandwidth in real-time.

1) QNetwork

The QNetwork is a neural network specifically developed to forecast the quality (Q-values) of executing particular actions in specified states within an environment. It is a fundamental component of Q-learning, which is a type of reinforcement learning. The architectural design consists of multiple fully connected layers. These layers, also referred to as dense layers, establish connections between each neuron in one layer and all neurons in the subsequent one. This enables the network to acquire intricate patterns within the data.

Functionality:

- Input: The current condition of the environment.

Cloud resources pertains to the present state or arrangement of the resources in the cloud.

- Output: The output consists of Q-values assigned to each action within the action space. These numbers indicate the anticipated usefulness of each action based on the current condition, which helps in choosing the most efficient action to optimize resource utilization.

2) TargetQNetwork

The TargetQNetwork is intricately connected to the QNetwork but fulfills a slightly distinct objective. It is employed to predict the future value of actions, offering a consistent objective for the QNetwork to acquire knowledge from. The TargetQNetwork possesses an identical architecture to the QNetwork, however with distinct weights.

Functionality:

- Periodically Updated: The weights of the TargetQNetwork are periodically updated with the weights from the QNetwork. This update is less frequent to provide stability to the learning process.
- Output: Input: The next state of the environment, which is the state following the current action.
- Output: Output: Q-values for each action in the action space, but calculated for the next state. These values are used to compute the target Q-value in the Q-learning update rule

B. DDQN Agent Class

The DDQN Agent Class is a notable breakthrough in the realm of reinforcement learning, specifically in handling intricate decision-making tasks such as optimizing cloud resources. It improves upon the original Deep Q-Network (DQN) by implementing a crucial adjustment that specifically tackles the underlying problem of overestimation bias in the DQN's methodology. This improvement not only improves the precision of the learning process but also makes the training phase more stable, resulting in a DDQN model that is more resilient and dependable for real-world applications. The core of DDQN is its dual-network structure, which utilizes two separate but interconnected neural networks - the QNetwork and the TargetQNetwork - both of which are crucial in the agent's learning and decision-making process. The main advancement of the DDQN Agent Class is its strategy of segregating the action selection and action evaluation procedures, which effectively mitigates the issue of overoptimistic value estimates commonly encountered in single-network Q-learning methods. The QNetwork, which is in charge of action selection, interacts with the environment and obtains input in the form of state transitions and incentives. Meanwhile, the TargetQNetwork, which is kept separate from direct learning, offers a consistent set of Q-values that are used to compare and modify the outputs of the QNetwork. This division guarantees a more impartial assessment of actions, resulting in more efficient and logical decision-making. In addition, the DDQN utilizes other essential techniques such as action selection through an epsilon-greedy approach, computation of target Q-values, calculation of loss, optimization of the model, and regular updates of the TargetQNetwork. Every one of these elements has a crucial function in the agent's capacity to acquire knowledge and adjust to a constantly changing environment, rendering DDQN a potent instrument for jobs that demand advanced, data-based decision-making.

1) Initialization

The initialization step establishes the DDQN agent by defining crucial characteristics and configurations required for its functioning.

- State Dimensions: State dimensions refer to the magnitude or quantity of variables in the state representation of the environment.
- Action Dimensions: This term refers to the total number of potential actions that the agent is capable of taking.
- Learning Rate: Dictates the speed at which the agent acquires new knowledge. An increased rate may result in accelerated learning but can also induce instability.
- Discount factor: The discount factor (gamma) is a numerical value ranging from 0 to 1 that quantifies the significance of future rewards in the decision-making process of the agent.

The exploration parameter, also known as epsilon, regulates the equilibrium between exploration, which involves experimenting with new strategies, and exploitation, which entails utilizing existing knowledge. Usually begins with a high level and gradually decreases over a period of time. The QNetwork and TargetQNetwork are initialized, each with their own optimizers. These networks are essential for calculating Q-values.

State Space: The state space describes the system's current state:

- Current Time: Integer / float representing the current time step.
- Task Queue: List of dictionaries, each containing details of waiting tasks (task ID, duration, deadline, priority).
- Resource Availability: List indicating available resources and their usage.
- Running Tasks: List of dictionaries with details of currently running tasks (remaining time, allocated resources).

Action Space: The action space represents possible decisions the agent can make:

- Schedule Task: Choosing a task from the queue (e.g., action is the index of the task in the queue).
- Allocate Resources: Assigning resources to the selected task (e.g., action is the specific resource or resource configuration).
- Wait: Option to wait for better scheduling conditions.

Reward Function: The reward function guides the learning process by providing feedback:

- Task Completion: Positive reward for successfully completing tasks.
- Resource Utilization: Positive reward for efficient use of resources.
- Deadlines: Penalty for missing deadlines.
- Fairness: Reward or penalty based on fairness in resource allocation.
- Latency: Penalty for high latency in task completion

2) Methods for Training

- select action Method: This strategy is crucial for the agent's interaction with the environment. It determines the course of action to be taken at each individual phase. The epsilon-greedy technique is of utmost importance in this context. The agent has the option to either explore the environment by randomly selecting an action, or use its existing knowledge by choosing the action with the greatest estimated Q-value, as determined by the QNetwork. The likelihood of exploration is determined by the epsilon parameter. This equilibrium guarantees that the agent avoids being trapped in a suboptimal solution and consistently acquires knowledge about the surroundings. Usually, epsilon is adjusted to decrease progressively as the agent gains more knowledge about the environment, resulting in reduced random exploration and increased utilization of the learnt policy.
- Method for Computing q target Details of Computation: The goal Q-value is an essential element in the update rule of Q-learning. The target Q-value for a specific state-action combination is computed by adding the reward obtained from executing that action to the discounted Q-value of the optimal action in the subsequent state, as predicted by the TargetQNetwork. This approach guarantees the policy is consistently revised to align with a long-term optimal strategy, taking into account forthcoming incentives. By utilizing the TargetQNetwork for this computation, the approach circumvents the frequent updates that may arise from employing the QNetwork, hence offering a more consistent target for the QNetwork to acquire knowledge from.
- compute q loss Method Calculation of Loss: This technique computes the mean squared error loss by comparing the predicted Q values obtained from the QNetwork with the target Q-values obtained from the compute q target method. The loss is the discrepancy between the current estimated value of a state-action combination and the model's expected value for that pair. The calculated loss is utilized to modify the weights of the QNetwork. By reducing this loss, the QNetwork's predictions gradually improve over time, resulting in enhanced decision-making.
- optimize model Method: During this step, the QNetwork's weights are modified using gradient descent algorithms. The gradients are computed using the loss obtained from the compute q loss technique. This method is crucial for the agent's learning as it ensures that the anticipated Q-values are in line with the observed rewards and the long-term value of future states. Usually, this procedure is referred to as being executed after every individual action or a group of actions to guarantee the agent's policy is continuously learning and adapting.
- update target network Method: This method is responsible for synchronizing the weights of the TargetQNetwork with the weights from the QNetwork. The frequency of this update is often lower than that of the learning updates to the QNetwork. An established method involves updating the TargetQNetwork after a predetermined number of steps or episodes. The few changes to the TargetQNetwork contribute to the stabilization of the learning process. The TargetQNetwork ensures the provision of consistent target Q-values for the QNetwork's learning process. Its slower adaption serves to prevent detrimental feedback loops, wherein the QNetwork may swiftly alter its policy based on its own recent updates.

C. Training Loop

The optimize model function is crucial in the training and learning process of a DDQN (Double Deep Q Network) Agent. This strategy is closely linked to the manner in which the agent engages with its surroundings and acquires knowledge from its encounters. To have a full understanding of the optimize model method, let's analyze each step in the training loop individually. The training loop involves:

- **Selecting actions using the select action method :** During each iteration, the agent must determine the optimal course of action to undertake based on the current condition of the environment. The select action method is utilized to make this selection, commonly with an epsilon-greedy technique. This approach achieves a harmonious equilibrium between exploration, which involves experimenting with novel

actions, and exploitation, which entails utilizing the most optimal known action. This balance is crucial for the agent to efficiently acquire knowledge. The agent acquires useful data about the environment by choosing actions in different states and watching the resulting outcomes, which is essential for its learning.

- **Storing experiences in a replay buffer :** The gathered experiences (state, action, reward, next state) are kept in a data structure referred to as the replay buffer. The buffer plays a crucial role in stabilizing and enhancing the learning process. The replay buffer enables the agent to retain and reuse previous experiences, hence disrupting the association between consecutive learning samples. This results in enhanced and resilient learning.

- **Sampling from this buffer to compute Q-targets and Q_loss :** At regular intervals, the agent selects a set of events from the replay buffer. This batch is utilized to calculate the Q-values for both the present and subsequent states. The target Q-values are computed using the TargetQNetwork for the subsequent state, and these values are then contrasted with the predicted Q-values derived from the QNetwork for the present state-action pair. The disparity between these values is calculated as the Q-loss, which serves as a metric for assessing the degree to which the agent's predictions correspond with the observed events.

- **Regularly updating the TargetQNetwork to stabilize training :** Based on environmental interactions, the training loop updates the Policy and Value Networks using the train approach. Continuous agent performance improvement requires this loop. PPO-specific loss functions guide updates, highlighting the need to balance exploration and exploitation. The training loop helps the agent modify its policies and value estimations over numerous rounds, improving its capacity to traverse and adapt to complicated settings.

D. Model Implementation

Double Deep Q-Network (DDQN)

Neural Network Architectures:QNetwork : This neural network predicts the quality (Q-values) of actions in given states. It consists of multiple fully connected layers to capture intricate patterns in the data.

- TargetQNetwork: Similar in architecture to the QNetwork, it predicts the future value of actions, providing a stable target for the QNetwork to learn from. The weights of this network are periodically updated from the QNetwork to ensure stability.

Agent Initialization:

- State Dimensions: The number of variables in the state representation.
- Action Dimensions: The total number of possible actions the agent can take.
- Learning Rate: Dictates the speed of learning.
- Discount Factor (gamma): Balances immediate and future rewards.
- Epsilon (exploration parameter): Regulates exploration vs. exploitation, typically starting high and decreasing over time.

Training Methods:

- Select Action: Uses an epsilon-greedy strategy to balance exploration and exploitation.
- Compute Q-targets: Combines immediate rewards with discounted Q-values from the TargetQNetwork to compute target Q-values.
- Compute Q-loss: Calculates the loss between predicted Q-values from the QNetwork and target Q-values.
- Optimize Model: Updates QNetwork weights using gradient descent based on the computed loss.
- Update Target Network: Periodically synchronizes weights from the QNetwork to the TargetQNetwork.

E. Evaluation Methodology

1) Virtual Instances: Ten instances with varying numbers of VCPUs to represent different computational capacities.
2) Job Arrival Rates: Mean job arrival rates varied between 10 and 30.
3) Job Types: Balance between high CPU and high I/O instances, with half the jobs being I/O-intensive and half computation-intensive.

F. Scalability and Real-World Applicability

Scalability

- The DDQN model demonstrated robust performance under varying workloads, including different job arrival rates and job types.
- The use of neural networks allows for handling large and complex state-action spaces, making the approach scalable to larger cloud environments.

Real-World Applicability

- The model's ability to adapt to dynamic and unpredictable cloud environments highlights its potential for real-world applications.

- By consistently outperforming traditional scheduling algorithms in cost, success rate, and response time, the DDQN model proves to be a practical solution for efficient task scheduling in cloud computing.

4. Experimental Analysis

The experimental setup has ten virtual instances with different numbers of Virtual CPUs (VCPUs). These instances' VCPU counts vary, which affects their computational power. Table I completely lists each instance's VCPU levels and other relevant data. We assumed that a specific instance's computational efficiency is directly proportional to its VCPU count to match the simulated environment with practical settings. Each VCPU receives 1000 compute units to normalize comparisons.

A. Experimental Setup and Evaluation

1) Simulations:

- The effectiveness of the DDQN approach was measured using simulations.
- The experimental setup included ten virtual instances with varying numbers of Virtual CPUs (VCPUs), representing different levels of computational power.

2) Performance Metrics:

- The proposed model evaluated performance under varying mean job arrival rates (10 to 30).
- The evaluation included metrics such as cost, success rates, and average response time for different scheduling algorithms, including Random, Round-Robin, Earliest, and DDQN.

3) Comparative Analysis:

- The results were compared to other scheduling methods like Random, Round-Robin, and Earliest.
- The comparison highlighted the advantages of the DDQN algorithm in terms of lower costs, higher success rates, and better average response times across different job arrival rates.

4) Different Job Types:

- The performance was also evaluated with different percentages of computing-intensive and I/O-intensive jobs.
- The DDQN algorithm consistently demonstrated lower average response times and higher success rates, indicating its efficiency in handling various types of workloads.

5) The results are compared based on :

- Cost: The DDQN algorithm consistently showed lower costs compared to Random, Round-Robin, and Earliest algorithms under various conditions.
- Success Rates: DDQN had higher success rates, indicating better task completion efficiency.
- Average Response Time: DDQN achieved lower average response times, reflecting its ability to schedule tasks more efficiently.

B. Performance evaluation under varying mean job arrival rates

This study measures the efficiency of different scheduling algorithms in handling the inflow of jobs at varying mean arrival rates. The experiments are performed with mean job arrival rates between 10 and 30, as this range was sufficient to showcase the clear advantages of the proposed algorithm. However, increasing the arrival rates to higher, like 40, 50, or even 100, will not add any significance to the findings. Based on the works in the literature, the computing power needed for each task is modeled using the normal distribution. The timing of job arrivals follows a Poisson distribution, delivering a realistic testing scenario. The virtual environment is evenly split between high CPU and high I/O instances. Similarly, the job types are balanced, with half being I/O intensive and the other half computation intensive.

The Figure 1 compares mean job arrival rates for Random, Round-Robin, Earliest, and DDQN methods. The Figure 1 shows each strategy's expenses at different job arrival rates. Similarly, the Figure 2 shows work scheduling algorithm success rates across mean job arrival rates. These success rates show how well each algorithm performs problems. The data shows fascinating algorithm tendencies and performance variances. Different algorithms manage tasks differently in work scheduling. The Random method has modest success rates, fluctuating but remaining stable throughout job arrival rates. The Round-Robin approach works moderately, with a steady success rate, especially in settings with a constant amount of job arrivals.

The Earliest algorithm outperforms others in the 10 to 25 job arrival rate range. In certain situations, its arrival-time efficiency is a strength. In this comparison, the DDQN algorithm shines out. It consistently performs well across all mean job arrival rates. This consistency shows its resilience and dependability in effectively handling incoming tasks, making it a tempting alternative for situations requiring a reliable and high-performing algorithm. Understanding these success rates helps choose the best scheduling method for system needs. Workload changes, job prioritizing, and effective task management affect algorithm selection. The data highlights the strengths and performance of each algorithm, with DDQN showing a consistently high success rate, making it a versatile option across job arrival circumstances.

The Figure 3 shows the mean job arrival rates for various scheduling methods at different job frequency. These rates indicate each algorithm's task scheduling and processing performance under varied workloads. Findings show trends across algorithms. Random Scheduling has

TABLE I. Type of Instance

Instance Type	Number of VCPUs
High I/O	1
High CPU	2
High CPU	4

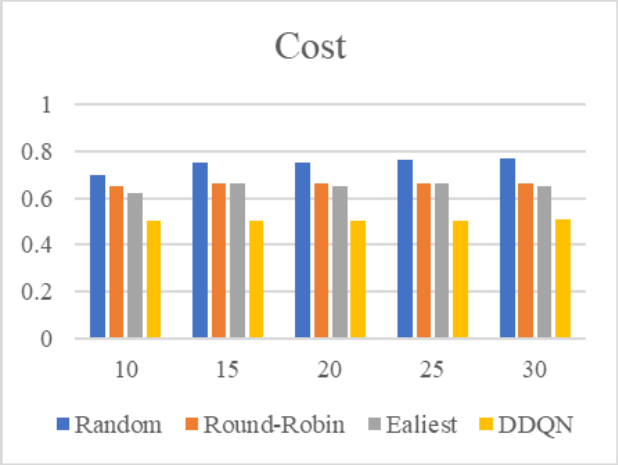

Figure 1. Cost Comparison of Different Scheduling Algorithms

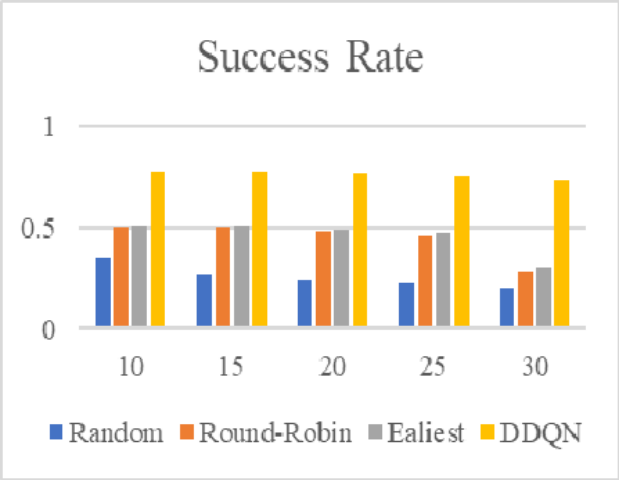

Figure 2. Comparison of Job Scheduling Algorithm Success Rates

somewhat greater mean job arrival rates than Round-Robin, Earliest, and DDQN at lower job arrival frequencies. As frequency rises, these algorithms' rates diverge further. Round-Robin responds to workloads by starting low and increasing rates moderately with task frequency. Earliest Scheduling has constant rates like Round-Robin but lower values. Meanwhile, DDQN consistently has lower mean job arrival rates across all frequencies, indicating a different method to managing incoming tasks than the other algorithms.

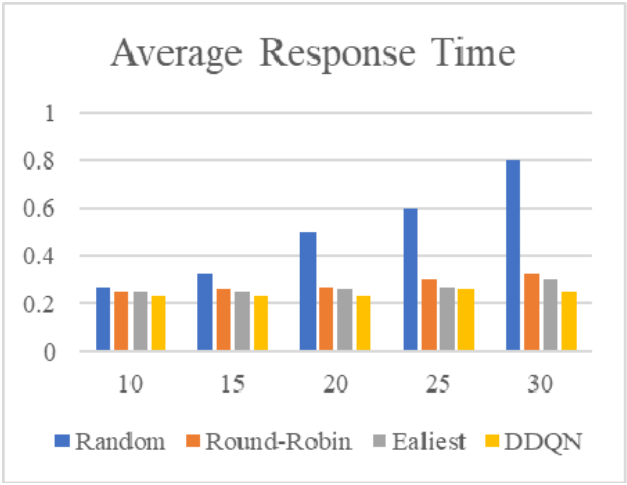

Figure 3. Average Response Time of the Different Task Scheduling algorithms

These findings illuminate scheduling algorithm efficiency and scalability. Since algorithm choice affects mean job arrival rates, DDQN regularly has lower rates. This suggests more efficient work management, particularly under stress. Random and Round-Robin systems have higher rates, especially as task frequencies grow, suggesting scalability issues. The steady rates of Earliest and DDQN algorithms suggest more consistent task processing, emphasizing the relevance of algorithm selection depending on workload and system performance.

C. Performance evaluation with different computing intensive jobs

Similar to the previous, in this experiment, along with the PPO, other scheduling techniques are tested with various computing-intensive job percentages. For this experiment, the mean job arrival rate is kept constant at 20, and computing intensive jobs percentage is varied from 10% to 90%. The infrastructure had 50% High CPU and 50% High I/O instances

In Figure 4, the depicted data illustrates the average response time of different scheduling algorithms under varying levels of system load, denoted by the random, round-robin, earliest deadline first (EDF), and deep double Q-network (DDQN) approaches. The table presents the average response time in seconds for each algorithm at load factors of 0.1, 0.3, 0.5, 0.7, and 0.9. Notably, the DDQN algorithm consistently demonstrates the lowest average response times across all load levels, showcasing its

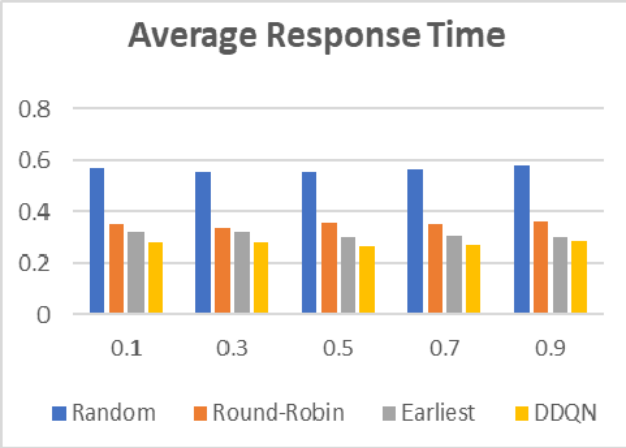

Figure 4. Average Response Time of the proposed algorithm with different computing intensive jobs

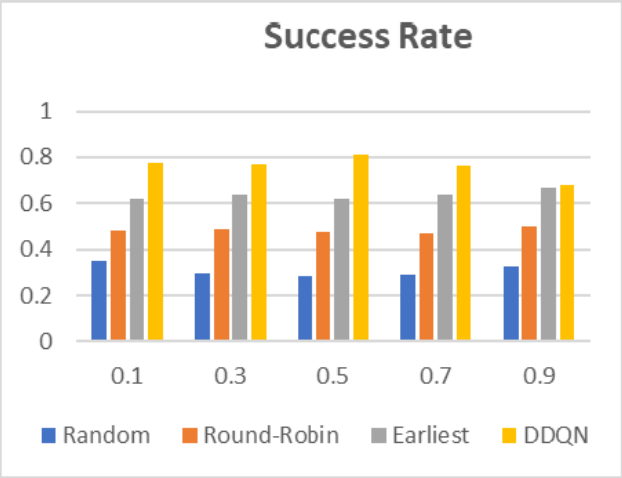

Figure 5. Success Rate of the proposed algorithm with different computing intensive jobs

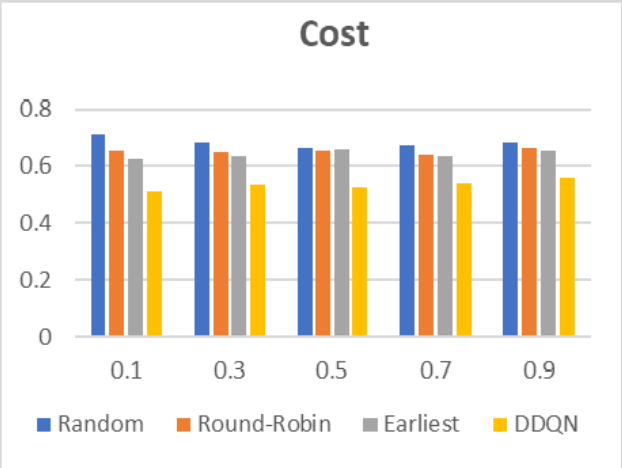

Figure 6. Cost of the proposed algorithm with different computing intensive jobs

efficiency in managing task scheduling in comparison to random, round-robin, and EDF algorithms. The values in each cell represent the average response time corresponding to the respective combination of scheduling algorithm and system load, providing valuable insights into the comparative performance of these strategies in diverse operating conditions.

Figure 5 displays a comparative examination of success rates for several scheduling methods, including Random, Round-Robin, Earliest, and DDQN (Deep Double Q-Learning Network), under different job success probability (0.1, 0.3, 0.5, 0.7, 0.9). The success rates indicate the effectiveness of each scheduling algorithm in accomplishing tasks within a certain probability range. Upon examination of the outcomes, it is clear that the DDQN algorithm constantly surpasses the other scheduling approaches in terms of task success rates, regardless of the varying probability of success.

Figure 6 presents a comparative examination of expenses related to several scheduling algorithms, including Random, Round-Robin, Earliest Deadline First (EDF), and Double Deep Q Network (DDQN). The table illustrates the expenses associated with different system loads, which range from 0.1 to 0.9. The expenses associated with each method under varying workloads are shown in the respective columns of the table. Significantly, when the system load rises, the expenses linked to the scheduling algorithms display diverse patterns. For example, the DDQN method regularly exhibits reduced costs for various workloads in comparison to the Random, Round-Robin, and EDF algorithms. The user's text is empty.

D. Performance evaluation with different I/O intensive jobs

This experiment examines the performance of different scheduling techniques, including the proposed PPO, under various high I/O instances. Here, the system is configured with a mean job arrival rate of 20 and Computing-intensive jobs at 50%, and the high I/O cases vary from 10% to 90%.

In Figure 7, the average reaction times of several scheduling algorithms, including Random, Round-Robin, Earliest, and DDQN (Deep Double Q-Learning Network), are shown for different task arrival rates. These rates are represented by the probabilities 0.1, 0.3, 0.5, 0.7, and 0.9. The table displays the mean reaction time (measured in a specific unit of time) for each scheduling method for various arrival rates. The Earliest algorithm consistently exhibits the lowest average reaction time for all arrival rates, indicating its effectiveness in lowering job completion durations. In contrast, the DDQN algorithm demonstrates strong performance, especially when faced with increased arrival rates, highlighting its efficacy in dynamic task scheduling circumstances.

Figure 8 displays a comparative examination of success

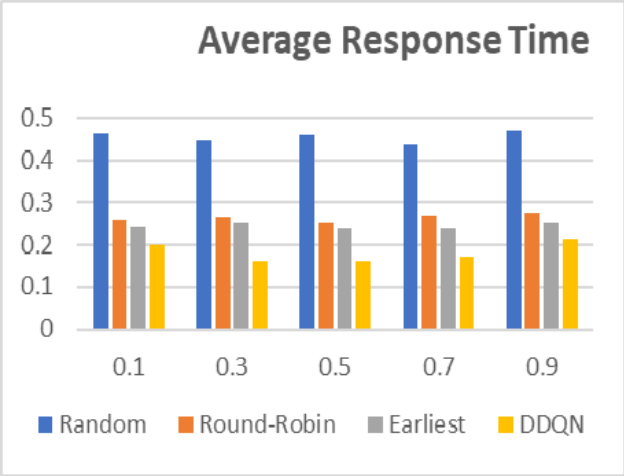

Figure 7. Average response time of the proposed algorithm with different I/O intensive jobs

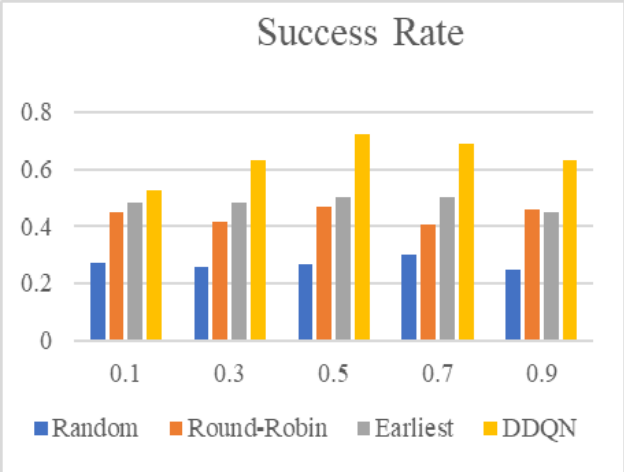

Figure 8. Success rate of the proposed algorithm with different I/O intensive jobs

rates for several scheduling algorithms, including Random, Round-Robin, Earliest Deadline First (EDF), and Double Deep Q-Network (DDQN). Every row corresponds to a unique experimental situation, while each column indicates the success rates attained by the corresponding scheduling algorithms in those instances. The success rates, represented by numerical figures, indicate the percentage of activities or processes that were successfully completed in a certain experimental context. For example, the Random algorithm achieves success rates between 0.25 and 0.3 in various settings, whereas the Round-Robin method has success rates ranging from 0.41 to 0.47. The Earliest Deadline First strategy has success rates ranging from 0.45 to 0.5, whereas the Double Deep Q-Network algorithm achieves success rates ranging from 0.525 to 0.725.

Figure 9 displays a comparative examination of several scheduling algorithms (Random, Round-Robin, Earliest,

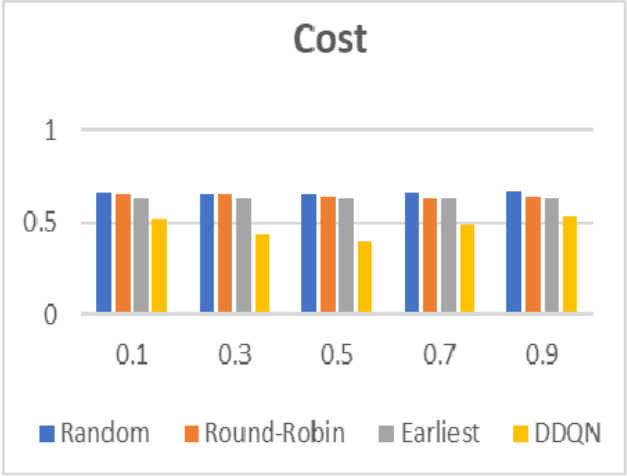

Figure 9. Cost of the proposed algorithm with different I/O intensive jobs

and DDQN) in terms of their costs for different task arrival rates (0.1, 0.3, 0.5, 0.7, and 0.9). The table presents the cost figures, which are monitored as performance indicators. The DDQN (Deep Double Q-Network) algorithm consistently exhibits reduced costs for all arrival rates in comparison to the Random, Round-Robin, and Earliest scheduling techniques. These findings indicate that DDQN is more effective in task management, leading to a greater level of optimization in resource allocation. The numbers in the table represent the comparative efficiency of each algorithm, where lower cost values imply superior performance in terms of job scheduling and execution.

The overall novelty of the proposed model is described as:

1) Enhanced Scheduling Efficiency:

The DDQN model consistently outperforms traditional scheduling algorithms such as Random, Round-Robin, and Earliest in terms of cost, success rate, and average response time. This demonstrates the model's capability to significantly improve scheduling efficiency in cloud environments, providing a robust solution for dynamic task management.

2) Real-World Applicability:

The model's effectiveness in handling various job arrival rates and types (I/O-intensive and computation-intensive) showcases its versatility and practical applicability. This proves the model's utility in real-world cloud deployments, where diverse and unpredictable workloads are common.

3) Scalability and Robustness:

The DDQN model shows stable performance and scalability, handling large–scale environments and complex workloads without degradation in performance. This establishes the model as a scalable solution for large cloud infrastructures, addressing a critical gap in current literature.

4) Adaptive Learning and Continuous Improvement:

The continuous learning mechanism of the DDQN model allows it to adapt and improve its scheduling policies over time based on real–time data. This highlights the model's potential for long–term deployment in dynamic environments, where continuous adaptation is crucial for maintaining optimal performance.

5) Reduction in Overestimation Bias:

By implementing the DDQN approach, the study demonstrates a reduction in the overestimation bias that is common in standard DQN algorithms. This leads to more accurate Q–value predictions and, consequently, more effective decision–making in task scheduling.

5. Conclusions

These experimental results clearly show that the DDQN technique performs well under balancing. DDQN is effective, responsive, and cost-effective when evenly distributed instance types. The DDQN model outperformed Random, Round-Robin and Earliest models in success rate, average response time, and cost in all three test conditions. DDQN model adapted nicely to varied operational settings, such as job arrival rates, computation intensive jobs, and high I/O instance jobs. Despite operational changes, the DDQN model demonstrated its stability and potential for complicated, real-time job scheduling and resource management scenarios. The tests show the ability of the DDQN model to optimize job execution in various dynamic computing contexts.

References

[1] S. Mangalampalli, G. R. Karri, M. Kumar, O. I. Khalaf, C. A. T. Romero, and G. A. Sahib, "Drlbtsa: Deep reinforcement learning based task-scheduling algorithm in cloud computing," *Multimedia Tools and Applications*, pp. 8359–8387, 2024.

[2] X. Wang, L. Zhang, Y. Liu, and Y. Laili, "An improved deep reinforcement learning-based scheduling approach for dynamic task scheduling in cloud manufacturing," *International Journal of Production Research*, pp. 4014–4030, 2024.

[3] M. Shiva Rama Krishna and S. Mangalampalli, "A novel fault-tolerant aware task scheduler using deep reinforcement learning in cloud computing," *Applied Sciences*, pp. 12 015–12 020, 2023.

[4] S. Iftikhar, M. M. M. Ahmad, S. Tuli, D. Chowdhury, M. Xu, S. S. Gill, and S. Uhlig, "Hunterplus: Ai based energy-efficient task scheduling for cloud–fog computing environments," *Internet of Things*, p. 100667, 2023.

[5] K. Li, Z. Peng, D. Cui, and Q. Li, "Sla-dqts: Sla constrained adaptive online task scheduling based on ddqn in cloud computing," *Applied Sciences*, p. 9360, 2021.

[6] Z. Chen, J. Hu, X. Chen, J. Hu, X. Zheng, and G. Min, "Computation offloading and task scheduling for dnn-based applications in cloud-edge computing," *IEEE Access*, pp. 115 537–115 547, 2020.

[7] K. Kumaran and E. Sasikala, "An efficient task offloading and resource allocation using dynamic arithmetic optimized double deep q-network in cloud edge platform," *Peer-to-Peer Networking and Applications*, pp. 958–979, 2023.

[8] S. Mangalampalli, G. R. Karri, M. Kumar, O. I. Khalaf, C. A. T. Romero, and G. A. Sahib, "Drlbtsa: Deep reinforcement learning based task-scheduling algorithm in cloud computing," *Multimedia Tools and Applications*, pp. 8359–8387, 2024.

[9] K. Kang, D. Ding, H. Xie, Q. Yin, and J. Zeng, "Adaptive drl-based task scheduling for energy-efficient cloud computing," *IEEE Transactions on Network and Service Management*, pp. 4948–4961, 2021.

[10] M. Sharma and R. Garg, "An artificial neural network based approach for energy efficient task scheduling in cloud data centers," *Sustainable Computing: Informatics and Systems*, p. 100373, 2020.

[11] S. Swarup, E. M. Shakshuki, and A. Yasar, "Task scheduling in cloud using deep reinforcement learning," *Procedia Computer Science*, pp. 42–51, 2021.

[12] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Computing and Applications*, pp. 1531–1541, 2020.

[13] H. Hou, S. N. A. Jawaddi, and A. Ismail, "Energy efficient task scheduling based on deep reinforcement learning in cloud environment: A specialized review," *Future Generation Computer Systems*, 2023.

[14] T. Dong, F. Xue, C. Xiao, and J. Li, "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment," *Concurrency and Computation: Practice and Experience*, p. e5654, 2020.

[15] Z. Zhao, X. Shi, and M. Shang, "Performance and cost-aware task scheduling via deep reinforcement learning in cloud environment," in *International Conference on Service-Oriented Computing*. Springer, 2022, pp. 600–615.

[16] Y. Ping, Y. Liu, L. Zhang, L. Wang, and X. Xu, "Sequence generation for multi-task scheduling in cloud manufacturing with deep reinforcement learning," *Journal of manufacturing systems*, pp. 315–337, 2023.

[17] S. Muniswamy and R. Vignesh, "Dsts: A hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment," *Journal of Cloud Computing*, p. 33, 2022.

[18] Q. Li, Z. Peng, D. Cui, J. Lin, and H. Zhang, "Udl: a cloud task scheduling framework based on multiple deep neural networks," *Journal of Cloud Computing*, p. 114, 2023.