Transactions on Emerging Telecommunications Technologies

WILEY

RESEARCH ARTICLE

Actor-Critic Optimization Based Adaptive Task Scheduling Using Deep Double Dueling Q Network for Heterogeneous Cloud Environments

C. Felsy[1] | R. Isaac Sajan[2]

[1]Department of Artificial Intelligence and Data Science, Ponjesly College of Engineering, Nagercoil, Tamilnadu, India | [2]Department of Electronics and Communication Engineering, Ponjesly College of Engineering, Nagercoil, Tamilnadu, India

Correspondence: C. Felsy (felsyc395@gmail.com)

Received: 4 January 2025 | **Revised:** 18 April 2025 | **Accepted:** 1 June 2025

Funding: The authors received no specific funding for this work.

Keywords: dynamic workload | real-time resource optimization | resource utilization | virtual machines

ABSTRACT

In the rapidly evolving domain of cloud computing, the efficient scheduling of dependent tasks is critical for optimizing resource utilization and achieving key objectives such as minimizing latency and maximizing throughput. This paper presents the Actor-Critic Optimization based Adaptive Task Scheduling using Deep Double Dueling Q Network (ACTS-D3QN) in heterogeneous cloud environments enhances cloud task scheduling by incorporating advanced machine learning and optimization techniques. The D3QN framework is structured as an actor-critic model, where the actor component handles task scheduling and resource allocation, and the critic component refines these schedules. The actor component of D3QN integrates a Proportional Integral Derivative (PID) Controller for adaptive scheduling, ensuring real-time optimization of resource allocation while adhering to strict deadlines and dynamically managing workloads. Additionally, the system introduces a Dynamic Data Placement Algorithm with Predictive Caching (DDPPC), aimed at improving data locality and minimizing data transfer times. To balance operational costs with performance, a Modified NSGA-III algorithm is employed in the critic component of D3QN for multi-objective optimization. Furthermore, constraint programming is leveraged for efficient task-to-resource matching. Experimental results demonstrate that the ACTS-D3QN method achieves significant improvements, including a 22.14% reduction in makespan and a 20.0% increase in throughput, thereby validating its effectiveness in dynamic cloud environments.

1 | Introduction

In the ever-evolving ecosystem of cloud computing, efficient scheduling of dependent tasks plays a pivotal role in ensuring optimal resource utilization, meeting service-level agreements (SLAs), and maintaining seamless operational workflows [1, 2]. Unlike independent task scheduling, dependent task scheduling requires the consideration of task interdependencies—where the execution of one task is contingent on the successful completion of others. Such scenarios are prevalent in complex computing domains including large-scale data analytics pipelines [3], scientific workflows [4], and distributed systems management [5]. In these applications, tasks often exhibit varying execution priorities, deadlines, and levels of criticality, necessitating intelligent scheduling mechanisms that can handle both resource constraints and temporal dependencies effectively [6, 7].

© 2025 John Wiley & Sons Ltd.

Despite its criticality, dependent task scheduling remains one of the most challenging and unsolved problems in cloud environments [8]. Existing scheduling strategies often fall short in addressing the trifecta of cloud task scheduling: respecting task precedence constraints, ensuring timely completion of jobs, and optimizing the use of heterogeneous cloud resources [9, 10]. These challenges are exacerbated by the inherent dynamicity and unpredictability of cloud workloads, where resource availability and demand can fluctuate in real-time due to multi-tenant environments and service elasticity [11, 12]. Moreover, factors such as data locality optimization [13], resource fragmentation, and the heterogeneity of virtualized infrastructures [14, 15] further complicate the design of effective scheduling mechanisms. Traditional approaches, including heuristic-based methods like the Earliest Deadline First (EDF) [16] and the Critical Path Method (CPM) [17], provide quick solutions by prioritizing tasks based on static criteria. However, these methods lack the flexibility to adapt to runtime variations in workload characteristics, often resulting in inefficient scheduling decisions and resource underutilization under dynamic cloud settings [18–20].

To address the shortcomings of heuristics, machine learning, and more specifically, reinforcement learning (RL), has been introduced into the task scheduling paradigm. Q-learning, a model-free RL technique, has been employed to learn optimal task scheduling policies through environment interaction, evaluating long-term cumulative rewards for each decision [21–23]. Although Q-learning enhances adaptability and allows for real-time decision-making, it is not without limitations. Its application in real-world cloud systems is constrained by computational overhead, slow convergence rates, and scalability issues, especially when operating in large, complex, and heterogeneous cloud infrastructures [24, 25].

Additionally, multi-objective optimization (MOO) methods such as Genetic Algorithms (GA) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) have gained attention for their ability to find a trade-off between conflicting objectives—such as minimizing execution time, maximizing resource efficiency, and balancing energy consumption. However, these methods also suffer from increased computational complexity, particularly in high-dimensional decision spaces, and may struggle to deliver real-time responses in large-scale environments.

This paper presents next-generation advancements in cloud task scheduling by integrating state-of-the-art methods to optimize performance across various domains. The motivation for this research arises from the limitations of existing scheduling approaches, which struggle to balance execution efficiency, resource utilization, and cost-effectiveness in dynamic cloud environments. The proposed ACTS-D3QN architecture is designed to tackle the complexities of task scheduling and resource allocation through an actor-critic model. In this framework, the actor efficiently manages task distribution based on real-time resource availability, ensuring optimal task execution, while the critic continuously refines these schedules using a MOO approach. By dynamically balancing costs and performance, the ACTS-D3QN architecture adapts to workload variations, improving scheduling efficiency, resource management, and overall system reliability in cloud environments.

- Designing PID Controller-Based Adaptive Scheduling ensures real-time resource allocation while adhering to strict deadlines and adapting to dynamic workloads. The PID Controller continuously fine-tunes resource allocation based on system variations, while D3QN enhances decision-making by evaluating current conditions and optimizing scheduling strategies. This integration maintains deadline adherence, improves scheduling efficiency, and optimizes cloud resource management.
- Dynamic Data Placement Algorithm with Predictive Caching (DDPPC) optimizes real-time data placement to enhance data locality and reduce transfer times. By analyzing historical data access patterns, DDPPC proactively caches frequently used data, ensuring its availability when needed. This minimizes latency and network congestion, leading to improved data access speed and overall system performance, particularly for large-scale cloud storage environments.
- To enhance MOO, a Modified NSGA-III integrates the strengths of NSGA-III and Parallel Genetic Algorithm (PGA). NSGA-III efficiently handles conflicting objectives such as minimizing operational costs while maximizing performance, while PGA accelerates solution convergence by leveraging parallel computation. This hybrid approach dynamically adapts to cloud workload complexities, achieving an optimal balance between cost and performance while ensuring efficient load distribution.
- Constraint programming is employed to manage resource heterogeneity effectively, ensuring that tasks are allocated to the most appropriate resources. High-performance nodes handle computationally intensive tasks, while lower-capability nodes manage less demanding workloads, optimizing resource utilization and system efficiency. This method enhances overall task execution, reduces processing bottlenecks, and ensures balanced workload distribution in heterogeneous cloud environments.

The organization of the rest of the paper is as follows: Section 2 presents related works; Section 3 introduces the proposed ACTS-D3QN method; Section 4 presents the experimental results and their analysis. Finally, Section 5 draws conclusions, in which will be summarized the main takeaways and some promising directions for further research.

2 | Related Works

This section presents a comprehensive review of recent task-dependent scheduling strategies in cloud computing environments, focusing on techniques that address challenges such as resource allocation, load balancing, fault tolerance, and energy efficiency.

Mangalampalli et al. [26] proposed deep Q learning-based scheduling for optimal allocation of tasks. It focuses on making critical decisions based on both past and upcoming tasks to be performed. Initially, the system receives requests from various applications, and each task's priority is determined by the task manager. These priorities are then fed into the DQN model

for further processing. Subsequently, the tasks are scheduled and placed in the execution queue. Concurrently, the system continuously monitors upcoming requests. The virtual machine (VM) manager allocates available VMs for each task, while the resource manager gathers resource information. Based on the data available to the VM manager and the DQN model, dynamic decisions are made to efficiently manage resources. Simultaneously, the execution queue progresses to execute the next task in line.

Alruwais et al. [27] proposed the farmland fertility algorithm to optimize makespan in resource scheduling, FFARS-MSO. Inspired by the strategy followed by farmers in choosing lands based on soil quality, this algorithm tries to optimize the makespan by performing load balancing and trying not to over utilize the available resources. Thereby, the strategy adopted will be distributing resources uniformly based on demand across hundreds of servers with numerous VMs. These VMs exhibit significant efficiencies regardless of whether the servers are homogeneous or heterogeneous. Despite these efforts, resource fragmentation remains a challenge in the scheduling process.

Mustapha and Gupta [28] presented a Density-Based Spatial Clustering to perform effective scheduling of tasks. The clustering technique is primarily utilized to reduce the search space, categorizing resources based on their utilization levels over varying time periods. The methodology operates in two main steps. Initially, the algorithm performs clustering by selecting an epsilon value and computing the distance value of core points. Core points are then elected based on which the locations of other points are determined. Borderline points are computed to establish connections between different clusters, while noisy points that do not belong to any cluster are identified and discarded. Secondly, for scheduling purposes, the min-max algorithm is employed to select the best cluster. The chosen cluster exhibits reduced execution time and delay, facilitating minimized search time for available resources.

Gupta et al. [29] introduced the network with the harmony search algorithm for the development of the system for fault awareness in resources allocation. The methodology begins by initializing both the task and the VM list, along with parameters such as the size of the harmony memory, rate of pitch adjustment, and bandwidth. The system looks for the best solution during training; it does so by assigning random VM ids to each task. In the case of scheduling, an ideal schedule is prepared based on the best solution; this is done by incorporating memory updates and a stopping condition. Later, each task is assigned a VM, and the complete schedule would be generated within the required time limit. During the process of error correction, weights of neurons are readjusted in order to generate the desired output with accuracy. A resource optimization problem is approached whereby fault tolerance will be achieved through continuous adjustments of the weights of the neural network.

Muneeswari et al. [30] proposed a technique that was based on the load balancing of VMs by a Bi-LSTM-based load balancer. First of all, it collects multiple tasks through a single task collector and forwards the tasks to the load balancer. The load balancer is equipped with Bi-LSTM, which then carries out the process of load balancing across the VMs. If it detects some imbalance in the network, it initiates migration of the VM by transferring task details to the balancer. The model will be optimized using genetic expression programming that includes the expression of genes denoted by head and tail. The schedules collected after optimization are transferred for execution to the VMs. It utilizes a hybrid of Bi-LSTM and GEP optimization techniques to achieve runtime workload balancing across VMs for efficient resource utilization.

Dubey and Sharma [31] proposed a hybrid multifaceted task scheduling. It was formed with two techniques, such as particle swarm and ant colony optimization. A set of steps is there to form the methodology. First of all, the users of the cloud submit different types of tasks with various QoS using public channels. A designed cluster model then collects such tasks. The cluster model maps the different tasks according to their necessary resources for execution. In this, an estimation about the required computational resources, memory, and other per-task needs is done. Various task manager-related analyses consider task characteristics, QoS requirements, and pricing considerations. All these help in deciding upon a correct distribution of the tasks among the various resources. A scheduler would assign tasks to heterogeneous resources based on the analysis it receives from the task manager. That ensures resources utilized are effectively used toward meeting QoS requirements while trying to minimize costs. Ultimately, the resource manager oversees the variable-priced VMs with their respective capacities of storage. It is aimed at ensuring that the VMs are used well and dynamically allocated to tasks with demand and unavailability.

Mangalampalli et al. [32] proposed gray wolf optimization for task scheduling with multi objectives. In this, the tasks initially submitted through the console of the cloud platform. The broker then further submits these tasks to the task manager on behalf of the users. The task manager verifies whether the tasks should be assigned to virtual available resources. Based on the SLA requirements, the cloud owner is advised to adopt multi-objective scheduling algorithms. These algorithms prioritize tasks based on multiple criteria, such as completion time, resource utilization, and cost optimization. Priorities for task scheduling are computed based on factors such as the unit cost of electricity at different data centers. Tasks with higher priority are scheduled first to optimize resource utilization and minimize costs.

Ali et al. [33] also proposed multi-objective Harris Hawks Optimization (MoHHOTS) while working on the task scheduling in cloud-fog computing; this particular algorithm optimized both transmission delay and energy consumption. MoHHOTS, leveraging the Harris Hawks Optimization's high convergence and its stochastic operators, effectively obtains a 25% improvement over existing methods in simulation tests toward the effective balancing of delay and energy trade-offs while diminishing the network overhead. However, being stochastic in nature, it faces variability in performance. Also, the model has its limitations in handling other objectives like transmission cost, security, and load balancing. To add to this, alternative metaheuristic algorithms can be integrated with advanced artificial intelligence techniques to enhance algorithm applicability further, which can also be a future direction of this work.

Ye et al. [34] proposed the SMWDSA algorithm for effectively solving the NP-hard problem of stochastic multi-workflow scheduling within cloud environments. This was done in

TABLE 1 | Comparative analysis of intelligent task scheduling and resource allocation techniques in cloud environments.

Author	Technique/algorithm	Objective	Limitations/remarks
Mangalampalli et al. [26]	Deep Q Learning-based Scheduling	Optimal task allocation	Requires continuous data monitoring and dynamic decision-making complexity
Alruwais et al. [27]	Farmland Fertility Algorithm (FFARS-MSO)	Minimize makespan, load balancing	Resource fragmentation remains a challenge
Mustapha and Gupta [28]	Density-Based Spatial Clustering	Efficient task scheduling	Dependent on epsilon value selection for clustering accuracy
Gupta et al. [29]	Harmony Search Algorithm with Neural Network	Fault-aware resource allocation	Neural network tuning and fault-tolerance complexity
Muneeswari et al. [30]	Bi-LSTM Load Balancer + Genetic Expression Programming	Runtime workload balancing	Complexity in Bi-LSTM training and gene expression integration
Dubey and Sharma [31]	Hybrid: Particle Swarm + Ant Colony Optimization	Multifaceted task scheduling	Multiparameter tuning and dynamic pricing complexity
Mangalampalli et al. [32]	Gray Wolf Optimization	Multi-objective task scheduling	Depends on accurate SLA data and task priority assignment
Ali et al. [33]	Multi-objective Harris Hawks Optimization (MoHHOTS)	Delay and energy consumption reduction	Performance variability due to stochastic nature; lacks handling of security and load balancing
Ye et al. [34]	SMWDSA Algorithm	Stochastic multi-workflows scheduling	Limited to single-cloud environments; does not address hardware faults
Cui et al. [35]	Multifactor MFO Algorithm	Task scheduling optimization	Implementation challenges and fine-tuning required; needs diverse knowledge transfer exploration
Ye et al. [36]	F-Ant Colony Optimization (F-ACO)	Workflow scheduling with cost and deadline focus	Balancing multi-objective scheduling under stochastic variability

three stages: preprocessing, scheduling, and feedback. SMWDSA handles task scheduling in such a way that each will be assigned a sub-deadline, it minimizes scheduling cost, and updates priorities for boosting. The experimental results showed superiority in performance regarding the state-of-the-art methods related to cost, resource utilization, success rate, and VM utilization. This algorithm does not take into consideration the failures of the executions of tasks due to hardware faults, and it also considers only a single-cloud system; hence, enhancing robustness and developing multicloud scheduling solutions are scopes for the future.

The MFO algorithm introduced by Cui et al. [35] improves the scheduling of cloud tasks through a multi-objective and multifactor optimization approach. It focuses on optimizing execution time, cost, and load balancing. By incorporating a strategy for knowledge sharing transfer, it enhances scheduling efficiency and outperforms other evolutionary algorithms. However, challenges may arise in implementing and fine-tuning optimization algorithms, and there is a need to explore diverse knowledge transfer methods to enhance performance across various cloud environments. Future studies will aim to enhance these algorithms and assess their effectiveness in handling multiple tasks simultaneously.

Ye et al. [36] proposed the F-ant Colony Optimization F-ACO algorithm, which tackles the challenge of efficiently scheduling workflows with cost considerations and deadlines specifically for IoT applications in cloud settings. This approach merges Colony Optimization ACO with a heuristic driven by costs, introducing a method for distributing deadlines and an adaptive task ordering system. These innovations aim to reduce scheduling expenses and improve convergence efficiency. The algorithm demonstrates performance in simulations surpassing advanced techniques when it comes to scheduling real-world workflows. Nonetheless, it has limitations including limited exploration of integrating learning and not yet addressing the complexities involved in multicloud scheduling. These aspects highlight potential avenues for future exploration (Table 1).

3 | Proposed ACTS-D3QN Methodology for Adaptive Task Scheduling

The data is fed into the model through a well-organized preprocessing pipeline, ensuring efficient model training and inference. Initially, raw data undergoes cleaning and normalization to maintain consistency across the dataset. Numerical features are scaled and categorical attributes are encoded. Once the preprocessing is complete, the data is partitioned into distinct sets for training, validation, and testing, which allows for performance evaluation. In the context of the proposed methodology, tasks, which include characteristics like dependencies, deadlines, and criticality levels, are generated and submitted to a task scheduler. These task attributes form the input state representation that feeds into the

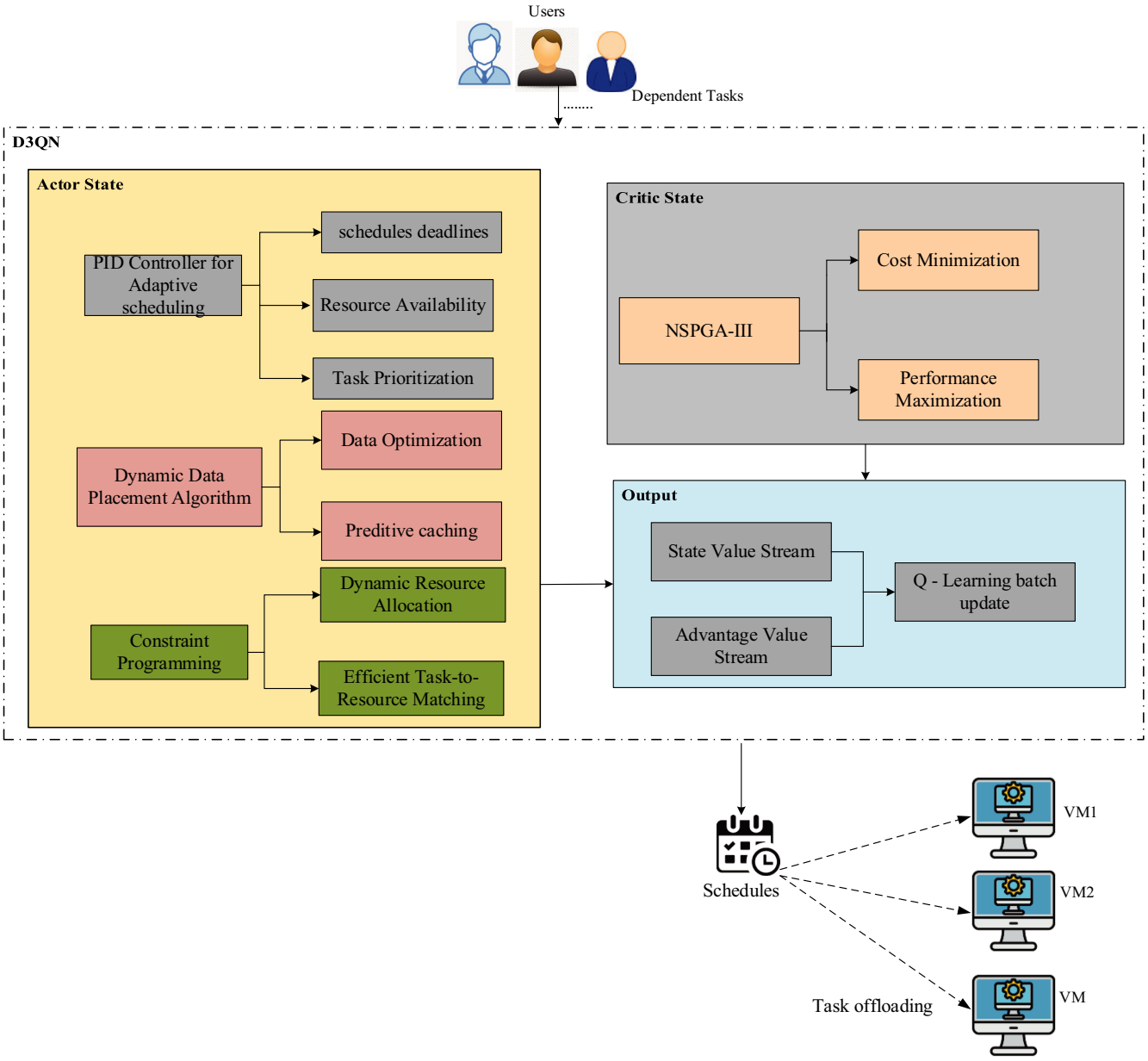


FIGURE 1 | Proposed architecture of ACTS-D3QN.

ACTS-D3QN architecture. This architecture incorporates a variety of scheduling modules to handle different aspects of task management, such as prioritizing tasks based on deadlines, dynamically adapting schedules to account for transient workloads, optimizing data locality for better resource utilization, managing the heterogeneity of resources, and balancing conflicting objectives. The model is then trained using deep Q-learning techniques that include prioritized experience replay, attention mechanisms, and MOO methods. These approaches help the D3QN model learn the best scheduling policies, aiming to maximize resource utilization while ensuring deadlines are met. The scheduling decisions generated by the ACTS-D3QN architecture, based on this processed data, help ensure that resources are efficiently allocated in the cloud environment. Overall, the structured preprocessing and input pipeline facilitate the seamless integration of data into the model, enabling it to guide the learning process and improve the performance of task scheduling and resource management. The entire architecture and methodology are further illustrated in Figure 1.

3.1 | System Model

The initial parameters define the most important points that can be used to enhance the task scheduling system in several aspects of scheduling, resource allocation, and time constraints conformance. In the dynamic cloud environment, using adaptive and predictive scheduling assigns and schedules tasks dynamically to the resources of cloud. Each task is represented by T_i, which has properties like execution time, deadline, and required resources. The scheduling environment is modeled with a dynamic resource pool R_j, resources are nonuniform to accommodate the different loads. The PID Controller-based Adaptive Scheduling works with the help of a PID controller to change the allocation strategy based on actual system performance and workload. The hybrid

auto-scaling scheme makes use of MOO for resource allocation. Utility function defines the various objectives in conflict such as resource acquisition, power consumption, and time of completion among others during the process of optimization.

The proposed task scheduling system initiates its work with an initialization phase where critical parameters are set up that are essentially required for its optimal performance on various aspects of scheduling, resource management, and deadline-related performances. Tasks allocated dynamically in the dynamic cloud environment will be scheduled among the available resources on the basis of adaptive and predictive scheduling principles. Each task is represented by T_i, characterized by parameters like execution time, deadline, and resource demand. The scheduling environment is designed with a dynamic resource pool R_j, where resources are heterogeneously distributed to handle different workloads. The PID Controller-based Adaptive Scheduling adjusts the allocation strategy based on real-time system performance and workload variations. Tasks are assigned to resources using a hybrid auto-scaling mechanism that leverages a MOO approach. The optimization process is guided by a utility function U_i that balances competing objectives such as resource utilization, energy consumption, and task deadline adherence.

$$U_i = \alpha \times E_i + \beta \times D_i + \gamma \times C_i \tag{1}$$

where E_i represents the energy consumption for task T_i, D_i denotes the degree of deadline adherence, C_i is the computational cost, and α, β, and γ are weights that balance the importance of energy, deadline, and cost considerations.

3.2 | ACTS-D3QN Model

Dataset Preprocessing and Model Input Pipeline: To ensure efficient model training and inference, the dataset undergoes a structured preprocessing pipeline. Initially, raw data is cleaned and normalized to maintain consistency. Numerical features are scaled using Min-Max normalization or standardization, while categorical attributes are encoded using one-hot encoding or embedding representations. Following preprocessing, the dataset is partitioned into training, validation, and test sets to facilitate performance evaluation. Mini-batch processing is employed to optimize memory usage and accelerate training convergence. Feature extraction techniques are applied to identify key attributes relevant to scheduling, resource allocation, and optimization. Additionally, time-series data is segmented using windowing techniques to enhance predictive capabilities in caching models. The processed dataset is then fed into the proposed framework, integrating PID Controller-Based Scheduling, DDPPC, Modified NSGA-III with PGA, and constraint programming.

Figure 1 illustrates a cloud computing task scheduling system that incorporates the D3QN architecture, which is based on a reinforcement learning model utilizing actor-critic methods. The D3QN model is designed to optimize the scheduling of dependent tasks commonly found in cloud environments. This architecture employs an actor-critic framework to handle the decision-making and evaluation processes effectively. In this setup, the actor is responsible for selecting or committing to an action based on the observed state of the environment. In contrast, the critic evaluates the actor's actions, providing feedback on the quality or value of the executed actions. The process begins with dependent tasks being fed into the D3QN model, where the Actor State assesses the scheduling tasks, resource availability, and task prioritization. The actor uses a PID Controller to perform adaptive scheduling, optimizing real-time task execution. The Dynamic Data Placement algorithm is then used to ensure data optimization and predictive caching, guaranteeing data availability for the tasks. Additionally, Dynamic Resource Allocation and constraint programming are employed to ensure that tasks are efficiently matched to available resources, enhancing overall system performance. On the other hand, the Critic State reviews the proposed schedules by assessing them based on two key objectives: cost minimization and performance maximization. This evaluation process is carried out using the NSGA-III algorithm, which is tailored for MOO, especially in scenarios where objectives conflict. The schedules that pass the critic's evaluation are then further refined using the Q-Learning batch process to optimize the task scheduling further. Once the schedules are optimized, they are offloaded to the VMs for actual execution, ensuring efficient management of cloud resources. This setup highlights the contribution of the improved task scheduler based on the Deep Distributed Double Q-learning Network, which learns to make optimal decisions in dynamic cloud environments. The D3QN enhances traditional Q-learning by incorporating a neural network to approximate the Q-function. The Q-value, representing the expected reward for taking an action in a given state, is updated according to the rule described in Equation (2). This combination of actor-critic methods, along with the advanced scheduling algorithms, ensures that the cloud system can adapt to the dynamic nature of cloud environments, making intelligent scheduling decisions that maximize resource utilization, minimize costs, and optimize performance.

$$Q(s,\alpha) \leftarrow Q(s,\alpha) + \alpha\left[r + \gamma \max_{a'} Q\left(s'\alpha'\right) - Q(a\alpha)\right] \tag{2}$$

In D3QN, Double Q-learning is applied to reduce overestimation bias by using two separate Q-networks. Equation (3) is used to select and evaluate actions:

$$Q_i(s,\alpha) \leftarrow Q_i(s,\alpha) + \alpha\left[r + \gamma Q_j(s', \arg\max_a, Q_i\left(s'\alpha'\right)) - Q_i(s,\alpha)\right] \tag{3}$$

Here, i and j represent different agents in the distributed system. This distributed approach allows the system to learn and adapt in real-time, dynamically reallocating resources based on current workload conditions, thereby optimizing task scheduling and resource utilization effectively.

D3QN can be combined with the actor-critic architecture to further enhance its decision-making capabilities in dynamic environments. In the actor-critic framework, the actor determines the best action α to take in a given state, using the policy $\pi(\alpha|s)$. The actor is then updated by adjusting the policy parameters θ in the direction that improves the expected reward by using Equation (4):

$$\nabla_\theta J(\theta) = E\left[\nabla_\theta \log \pi_\theta\left(a_t|s_t\right) Q(s_t, a_t)\right] \tag{4}$$

The critic uses Equation (5) the temporal difference (TD) error to update the Q-values:

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}) - Q(s_t a_t) \tag{5}$$

3.2.1 | Actor Framework in D3QN

The Actor in the D3QN architecture manages task scheduling by evaluating task dependencies, priorities, and deadlines. It assesses resource availability and prioritizes tasks to ensure efficient execution. Using a PID Controller, the Actor adapts scheduling decisions in real time. It also employs a Dynamic Data Placement Algorithm to optimize data availability and predictive caching, and applies Dynamic Resource Allocation and constraint programming to ensure efficient task-to-resource matching and handle complex constraints.

3.2.1.1 | Proportional Integral Derivative (PID) Controller-Based Adaptive Scheduling. The PID controller providing real-time adjustments to task scheduling based on system performance. The PID controller adjusts the scheduling of tasks to ensure that deadlines are met, even under varying conditions. The PID controller expression is given in Equation (6).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{6}$$

where $u(t)$ is the control variable (adjustments to task scheduling) and $e(t) = r(t) - y(t)$ is the error between the reference $r(t)$ (desired task completion time) and the actual output $y(t)$ (current task progress). K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively. The PID controller operates by calculating the error $e(t)$ and applying corrections based on three terms:

Error with workload variation:

$$e(t) = r(t) - y(t) + \kappa \cdot \Delta W(t)$$

where $\Delta W(t) = W(t) - W(t-1)$ is the workload change rate, and κ is a sensitivity factor.

Auto-tuning of gains:

$$K_p(t) = K_p^0 + \eta_p \cdot \frac{\partial U}{\partial e}, \qquad K_i(t) = K_i^0 + \eta_i \cdot \int \frac{\partial U}{\partial e} dt,$$
$$K_d(t) = K_d^0 + \eta_d \cdot \frac{d}{dt} \frac{\partial U}{\partial e}$$

where K_p^0, K_i^0, and K_d^0 are initial gains, η_p, η_i, and η_d are learning rates, and U is the utility.

Proportional Term ($K_p e(t)$): This term provides an immediate response to the error. In task scheduling, if the task completion time is behind the desired schedule, the proportional component increases the resource allocation proportionally to reduce the error.

Integral Term $K_i \int_0^t e(\tau) d\tau$: This term accounts for past errors. If tasks have consistently been late, the integral action increases resources over time to correct this accumulated delay, ensuring that the system compensates for persistent discrepancies.

Derivative Term $\left(K_d \frac{de(t)}{dt}\right)$: This term anticipates future trends by considering how quickly the error is changing. If tasks are increasingly delayed, the derivative component acts preemptively to allocate additional resources or adjust scheduling to prevent further delays.

In the context of cloud task scheduling, this PID-based approach dynamically adjusts the scheduling of tasks to minimize delays and ensure that tasks are completed within the specified deadlines, despite fluctuations in workload.

Adaptation to Workload Variations: The PID controller adapts to workload variations through a continuous feedback loop that monitors system performance metrics like CPU usage, memory consumption, and task completion rates and adjusts resource allocation in real-time. When a sudden workload spike occurs (e.g., an influx of high-priority tasks), the proportional term ($K_p e(t)$) immediately increases resource allocation to reduce the error $e(t)$ between the desired and actual completion times. For gradual workload increases, the integral term $K_i \int_0^t e(\tau) d\tau$ accumulates past errors, incrementally boosting resources to address persistent delays. The derivative term $\left(K_d \frac{de(t)}{dt}\right)$ anticipates trends, such as accelerating delays, and preemptively adjusts scheduling to stabilize performance. This dynamic interplay ensures responsiveness to both transient and sustained workload changes. Integrated with the D3QN framework, the PID controller's adjustments are further refined by the actor's learned policies, which optimize resource allocation based on long-term rewards, enhancing adaptability to complex, unpredictable cloud workloads.

3.2.1.2 | Dynamic Data Placement Algorithm With Predictive Caching. Especially in cloud environments, it is necessary to manage the data effectively to avoid latencies and optimize the system performance. The DDPPC is a stable method that aims at improving the data locality and the time taken to transfer data by combining real-time data placement with caching. In the next section, we discuss the elements of DDPPC with equations and elaborate explanation of each one of them. Dynamic Data Placement therefore entails the active identification of where exactly the data should be stored at any given time, so that the data is accessible at the right computation node or data center at any one time. In Equation (7) let D_i represent the dataset i and C_j represent the computation node or data center j. The objective of dynamic data placement is to minimize the transfer time T_{ij} from the data's current location to the target computation node:

$$Minimize\ T_{ij} = \frac{S(D_i)}{B_{ij}} \tag{7}$$

where $S(D_i)$ is the size of dataset D_i and B_{ij} is the bandwidth between the data's current location and the target data center C_j. Dynamic Data Placement evaluates the location of required datasets and initiates their transfer to the target computation node or data center where the analysis will occur. By minimizing the transfer time T_{ij}, the algorithm ensures that data is moved efficiently, reducing latency and enhancing performance. For instance, if the data is stored in Data Center B but the analysis is to be conducted in Data Center A, DDPPC ensures the data is transferred to Data Center A in real-time.

Predictive Caching: Predictive Caching uses historical data access patterns to anticipate future data needs and preemptively stores frequently accessed datasets at or near the computation node, reducing the need for future data transfers. Predictive caching can be modeled as an optimization problem where the objective is to maximize cache hit rate H by selecting the most frequently accessed datasets D_i for caching as given in Equation (8):

$$Maximize \ \ H = \sum_{i=1}^{n} P(D_i) \cdot I(D_i) \tag{8}$$

where $P(D_i)$ is the predicted probability of dataset D_i being accessed and $I(D_i)$ is an indicator function that is 1 if D_i is cached, and 0 otherwise. To predict data locality trends, the system continuously monitors data access requests across the cloud environment, collecting metrics such as access frequency, timestamps, and the computational nodes involved. These historical patterns are analyzed to estimate p_i for each dataset. This process leverages a lightweight statistical model such as exponential smoothing to prioritize recent accesses or integrates with the D3QN framework, where the actor component learns to predict access probabilities based on rewards from past caching decisions. The system generates a dynamic locality heatmap, a conceptual representation of "hot" (frequently accessed) and "cold" (rarely accessed) data regions, based on these probabilities. For example, datasets accessed repeatedly within short time windows are marked as "hot" and prioritized for caching. This predictive approach ensures that the system anticipates data needs proactively, adapting to evolving workload patterns in real-time.

The predictive caching component of DDPPC uses statistical models to analyze historical access patterns and predict which datasets are likely to be needed in the near future. These datasets are then preemptively cached at the computation node or a nearby data center. By maximizing the cache hit rate, predictive caching reduces transfer delays and ensures that data is readily available when needed, thus improving system efficiency.

Integration of Real-Time Data Placement With Predictive Caching: The integration of real-time data placement with predictive caching forms the core of DDPPC, providing a comprehensive approach to optimizing data locality and transfer times. The overall objective of DDPPC can be represented as a MOO problem. Equation (9) maximize both data transfer time T_{ij} and cache miss rate M:

$$Maximize \ \ \alpha T_{ij} + \beta M \tag{9}$$

where α and β are weighting factors that balance the importance of transfer time and cache miss rate, respectively and $M = 1 - H$, where H is the cache hit rate. Dynamic adjustment of storage allocation occurs as follows: when a dataset D_i is predicted to have a high p_i, the actor component initiates its transfer to the optimal computation node C_j if it's not already local, minimizing T_{ij}. Simultaneously, predictive caching allocates storage resources such as memory or SSD space by reserving cache slots for "hot" datasets, potentially evicting "cold" ones with lower p_i. This process is guided by real-time feedback from the PID Controller which fine-tunes cache size and bandwidth allocation based on latency deviations, and constraint programming which ensures storage assignments respect resource capacity constraints. For instance, if a high-performance node has ample storage, it may cache larger datasets for intensive tasks, while a lower-capability node caches smaller datasets for less demanding workloads. The D3QN critic evaluates these actions, refining the actor's policy to improve future decisions, ensuring adaptability to workload shifts. DDPPC simultaneously optimizes data placement and caching by balancing the goals of minimizing transfer time and maximizing cache efficiency. By dynamically placing data in the optimal location and preemptively caching likely-needed datasets, DDPPC ensures that data is both readily accessible and efficiently transferred, thereby reducing latency and improving overall system performance. DDPPC solves the challenges of data locality and resultant transfer times effectively in cloud computing environments. DDPPC provides an intelligent integration of real-time data placement with predictive caching to ensure that large datasets are available at the right place at the right time to avoid latency and hence enhance system efficiency.

3.2.1.3 | Managing Resource Heterogeneity through Constraint Programming. Resource heterogeneity is that there might be numerous types of resources in cloud computing systems, each having different characteristics and limitations. Constraint programming is used against this challenge in order to maximize matching tasks with the right resources. In constraint programming, it is important to consider and set each resource type's capacity and limitation within a given system. They range from computational capabilities to memory, network connectivity, among other related factors.

Let R_j represents resource j with capabilities $C_j = \{c_{j1}, c_{j2}, \ldots, v_{jm}\}$, where c_{jk} denotes the capability of resource j. Constraints for resource j can be expressed as given in Equation (10):

$$\forall_j, C_j = \left\{ c_{j1} \geq \text{threshold}_{j1}, c_{j2} \leq \text{limit}_{j2}, \ldots, c_{jm} = \text{value}_{jm} \right\} \tag{10}$$

where threshold_{jk} and limit_{jk} are the minimum and maximum acceptable values for capability k of resource j and value_{jm} is the fixed value for certain capabilities. These constraints define the operational boundaries of each resource. For example, a high-performance node might have a high CPU capability, while a low-power node may have constraints on energy consumption. By modeling these constraints, the system can accurately characterize the available resources, setting the stage for efficient task-resource matching.

Tasks within the system also have specific requirements that must be met by the resources to which they are assigned. These requirements are modeled in terms of the same capabilities defined for resources. Let T_i represent task i with requirement $R_i = \{r_{i1}, r_{i2}, \ldots, r_{in}\}$, where r_{ik} denotes k-th requirement of task i. Equation (11) is used to express matching condition:

$$\forall_i, \exists_j \quad \text{such that} \ \ r_{ik} \leq c_{jk} \forall k \tag{11}$$

Each task T_i has a set of requirements that must be satisfied by the resource R_j to which it is assigned. For instance, a computationally intensive task might require a resource with high processing power and memory capacity. The condition $r_{ik} \leq c_{jk}$ ensures that the selected resource can fully meet the task's requirements.

Task-Resource Matching via Constraint Programming: The core of the proposed solution is the matching of tasks to resources

using constraint programming. This involves solving a constraint satisfaction problem (CSP) where the goal is to assign each task to the most suitable resource, optimizing for system efficiency. The task-resource assignment problem can be formulated as given in Equation (12)

$$Minimize \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} \cdot d_{ij} \tag{12}$$

The objective function minimizes the total cost d_{ij}, which can represent factors such as energy consumption, latency, or resource utilization. The constraints ensure that each task is matched to a resource that meets its requirements and that resources are not overloaded. By solving this CSP, the system dynamically assigns tasks to the most appropriate resources, optimizing the use of diverse resources in real-time.

Dynamic Adjustment and Real-Time Adaptation: As the cloud environment changes, with varying task demands and resource availability, the system must adapt in real time. Constraint programming allows for dynamic reallocation of tasks to different resources as needed. The reallocation problem can be expressed as a dynamic CSP, where constraints are updated in response to changes in the environment by Equation (13).

$$\text{Update } C_j \text{ and } R_i \text{ as environmental changes} \tag{13}$$

After updating the constraints, solve the task-resource matching problem. With every change in the cloud environment—for example, an increase in workload or resource failure—the system updates the constraints and solves the task-resource matching problem again. Since the adjustment will be dynamic, tasks in such a context would always be assigned to the most suitable resources; this preserves efficiency in performance in case of changes. Constraint programming acts like an orderly and elastic foundation to handle resource heterogeneity in cloud computing by defining and modeling constraints over resources and tasks such that tasks can be matched dynamically with the best-suited resources. Constraint programming is a very strong tool for optimization of resource allocation in complex heterogeneous clouds, given its capability of real-time adaptation supported by intrinsic mathematical models.

3.2.2 | Critic Framework in D3QN

The actor-critic model in the system consists of two main components: the actor, which generates actions (in this case, scheduling decisions), and the critic, which evaluates the quality of those actions and refines the scheduling decisions over time. The critic plays a crucial role in enhancing the scheduling process by using a combination of the NSGA-III algorithm and the PGA, which together optimize cloud computing environments by handling conflicting objectives. The NSGA-III algorithm is specifically designed for MOO, making it ideal for addressing the cloud environment's complex trade-offs between goals like cost minimization and performance maximization. By generating a diverse set of Pareto-optimal solutions, NSGA-III ensures that the cloud system can explore a variety of optimal solutions, considering both performance and cost. This diversity of solutions allows the system to adapt to changing cloud environments by balancing conflicting objectives effectively. Meanwhile, the PGA enhances the computational efficiency of NSGA-III by leveraging parallelism. It divides the optimization process across multiple processors, enabling the evaluation of several subpopulations simultaneously. This parallelization accelerates the convergence process, allowing for faster refinement of the solution. The result is a more efficient and scalable approach to MOO. Beyond the use of NSGA-III, the critic further refines the schedules by evaluating the actions generated by the actor in real-time, providing continuous feedback. The critic's role is not limited to optimizing schedules based on static objectives; it also adapts to dynamic conditions in the cloud, such as varying workloads and resource availability. The critic works iteratively, improving the scheduling decisions by constantly refining the model's understanding of how to balance performance and cost effectively. In essence, the critic enhances the scheduling process by using the NSGA-III and PGA combination, ensuring that schedules evolve to meet the cloud environment's dynamic needs. This refined approach not only handles conflicting objectives more effectively but also ensures that resource usage is optimized for both cost and performance in real-time cloud operations. The flow of this process is illustrated in Figure 2, showing the interaction between the actor and critic and how they collaboratively optimize cloud resource allocation.

3.2.2.1 | NSGA-III for MOO. NSGA-III is a powerful evolutionary algorithm designed to handle MOO problems, especially those with more than two conflicting objectives. In this scenario, the primary objectives are minimizing operational costs and maximizing system performance. $C(x)$ is the operational cost associated with resource usage and $P(x)$ is the performance metric, which we seek to maximize. NSGA-III aims to find a set of Pareto-optimal solutions that balance these objectives. The optimization problem can be formulated by given in Equation (14):

$$Minimize\ C(x) \text{ and } Maximize\ P(x) \tag{14}$$

NSGA-III operates by sorting solutions into non-dominated fronts and applying a reference point-based selection mechanism to maintain diversity among the solutions. NSGA-III handles the trade-off between operational cost minimization and performance maximization very effectively. All these produce a diversified set of Pareto-optimal solutions that enable decision-makers to pick up an appropriate trade-off according to certain requirements of the cloud environment. This algorithm guarantees a good distribution in the objective space by its reference point-based selection mechanism for offering a wide range of options in balancing cost and performance.

3.2.2.2 | PGA for Computational Efficiency. PGA is used for making the process computationally more efficient by distributing the computational load in several processors. This being applied to large-scale optimization problems can be computation-intensive in a cloud environment.

Equation (15) is the fitness function $f(x)$ for a solution x evaluated in parallel across different subpopulations, which are processed independently:

$$f(x) = \sum_{i=1}^{n} \frac{P_i(x)}{C_i(x)} \tag{15}$$

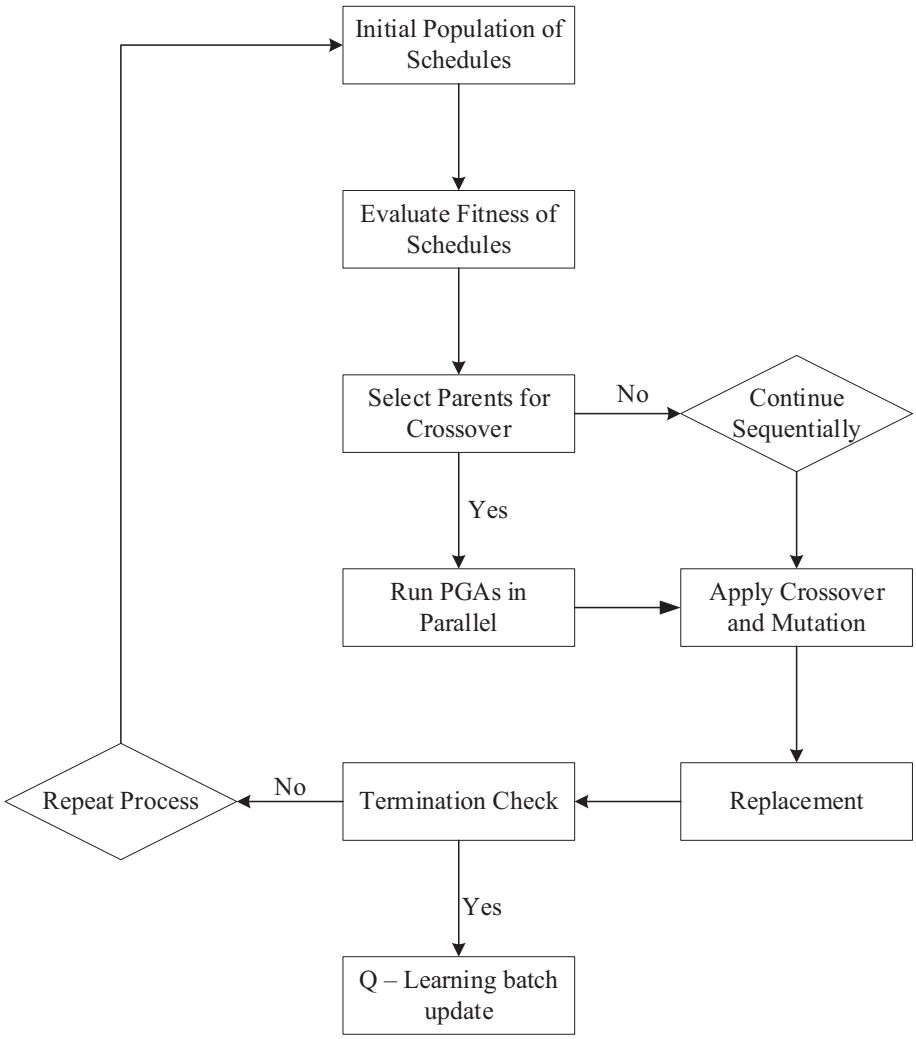


FIGURE 2 | Workflow of NSPGA-III.

where $P_i(x)$ and $C_i(x)$ are the performance and cost functions for subpopulation i, respectively and π is the total number of subpopulations. On the whole, the entire optimization process can be considered to perform several genetic algorithms in parallel, contributing to the global result. The PGA contributes by offering better scalability and increases the speed of the optimization process with the help of parallelism. By distributing the load, PGA ensures efficiency in the optimization process even if the complexity of the cloud environment increases. This is especially efficient in handling big data sets or when an optimization problem has several competing objectives whereby a better convergence to the optimal solutions can be achieved with speed.

3.2.2.3 | Integration of NSGA-III and PGA. The integration of NSGA-III and PGA provides a comprehensive solution for MOO in cloud computing environments. While NSGA-III focuses on managing conflicting objectives, PGA ensures that the optimization process is computationally efficient. The integrated optimization objective is denoted as given in Equation (16):

$$Minimize\ F(x) = [C(x) - P(x)] \tag{16}$$

Because PGA is parallel in nature, a number of subpopulations can run NSGA-III in parallel, following which the overall solution can be iteratively refined to convergence. Combining the strengths of NSGA-III with those of PGA gives a MOO platform which is not only robust but also quite efficient in handling most of the conflicting objectives with regards to computational resources. Such an integrated approach offers cost optimization and performance, dynamically changing with the whims of the cloud environment. The integration of NSGA-III and PGA within the Critic component establishes it as a versatile MOO framework. It ensures that the cost is optimized dynamically with performance and adapts to the complexities of the cloud environment. The Critic handles conflicting objectives efficiently by enhancing the computational efficiency of the algorithm; therefore, it ensures optimal utilization of cloud resources for the targeted performance outcomes.

4 | Experimental Details and Analysis

This section provides the outcomes of the experimental setup, including workload generation, performance evaluation of the proposed ACTS-D3QN scheduling technique, and comparisons with existing state-of-the-art task scheduling methods.

4.1 | Experimental Details

4.1.1 | Simulation Setup

The performance of the proposed ACTS-D3QN will be evaluated in a Python simulation environment that is designed to optimize task execution, dynamic adaptation of workload, data locality, and resource management. It also sets up a complex multicore system environment where several tasks with different priorities, resource requirements, and deadlines will be scheduled. Simulation will cover both static and dynamic workloads with a view to capturing real-world task demands and system conditions. There should be a central scheduling system for effective task management due to efficient resource allocation and proper execution of tasks on time. So, the key performance indicators are the task completion time, resource utilization, and system overhead, while the inclusion of energy efficiency and fault-tolerance effectiveness is also provided by Dynamic changes in task arrival rates and resource availability should be introduced during simulation to test the adaptability of the system and real-time performance. Scenarios within this full setup—from handling high-priority tasks to resource constraints—have given a full test of capability with regard to the method proposed herein. Parameters used for the review of the ACTS-D3QN method are shown in Table 2.

The simulation environment is configured with 500 VMs connected via 1000 network links, handling three types of tasks arriving at a rate of 50 tasks per second. The tasks are prioritized uniformly across high (1 h), medium (2 h), and low (4 h) priority levels. The system operates with four distinct resource types, where the resource allocation per type ranges from 10 to 100 units, while maintaining an energy efficiency threshold of 50 J. A redundancy of 10% is applied, meaning each task has one backup, and the system accommodates 10 000 tasks using a total of 1000 resources. The system dynamically adjusts every 30 s to optimize performance, with each task targeted to complete within 200 ms and consuming 50 J of energy. The system operates at 85% resource utilization, with an additional 10% overhead, over a total duration of 1 h. For learning configurations, an Adam optimizer is used with a tuned learning rate (α) for stability, alongside alternative optimization algorithms such as RMSprop or SGD. Training runs for 100 epochs with a batch size of 64, applying MSE loss for regression tasks and Cross-Entropy loss for classification tasks. Feature extraction considers performance metrics like CPU usage, memory consumption, execution time, I/O operations, and network latency. The Epsilon-Greedy policy is applied to balance exploration and exploitation, with a discount factor (γ) of 0.9. Regularization is handled through L2 regularization, and a dropout rate of 0.5 is implemented to prevent overfitting.

TABLE 2 | Parameters used to assess proposed ACTS-D3QN.

Parameter	Value
VMs	500
Number of Links	1000
Number of task types	3
Task arrival rate	50 tasks/s
Task deadline distribution	Uniform
Task deadline (high priority)	1 h
Task deadline (medium priority)	2 h
Task deadline (low priority)	4 h
Number of resource types	4
Resource allocation max	100 units
Resource allocation min	10 units
Energy efficiency threshold	50 J
Redundancy percentage	10%
Number of backups	1 backup per task
Number of tasks	10 000
Number of resources	1000
Adjustment interval	Every 30 s
Task completion time	200 ms
Energy consumption	50 J
Overhead	10%
Resource utilization	85%
Total duration	1 h
Learning rate (α)	Adam optimizer
Batch size	64
Number of epochs	100
Optimization algorithm	Adam, RMSprop, or SGD
Loss function	MSE (for regression), Cross-Entropy (for classification)
Feature extraction parameters	CPU usage, memory, execution time, I/O, network latency
Exploration-exploitation policy	Epsilon-Greedy
Discount factor (γ)	0.9
Dropout rate	0.5
Regularization	L2

4.1.2 | Dataset Description

The Google Cloud Jobs (GoCJ) dataset [37] is a large-scale collection of cloud computing workload traces, containing an average of 670 000 traces and approximately 30 million tasks executed across over 12 000 VMs within a 30-day period. This dataset provides a detailed representation of real-world cloud workload characteristics, enabling the evaluation of scheduling efficiency, resource management, and system performance. It includes essential task execution attributes such as start and end times, allowing for an in-depth analysis of task duration, execution patterns, and system responsiveness. Additionally, it offers resource utilization metrics, including CPU usage percentage, maximum CPU consumption, and memory-related parameters such as canonical memory usage, allocated memory, and peak memory consumption. Furthermore, the dataset provides insights into disk I/O performance, capturing disk input/output duration, local drive space usage, and peak disk I/O time, which are critical for assessing data access efficiency and identifying potential storage bottlenecks. Memory management data, such as total and uncharted page cache, also plays a key role

in understanding memory-intensive workloads and optimizing memory allocation strategies. Table 3 presents the dataset split of the GoCJ traces used for model training, validation, and testing.

4.1.3 | Performance Metrics

The metrics for evaluating the ACTS-D3QN method is given below:

Makespan (M): The total time required to complete all tasks from the start to the finish of the scheduling process is calculated in Equation (17).

$$\text{DI} = \frac{1}{R}\sum_{r=1}^{R}\frac{|\,U_r - \text{Average Usage}\,|}{\text{Average Usage}} \tag{17}$$

where T_i is the completion time of task i, and n is the total number of tasks.

Average Resource Utilization Ratio (ARUR): The ratio of the total resource usage to the total available resource capacity is calculated by Equation (18).

$$\text{ARUR} = \frac{1}{R}\sum_{r=1}^{R}\frac{U_r}{C_r} \tag{18}$$

where U_r is the usage of resource r, C_r is the total capacity of resource r, and R is the total number of resources.

Throughput (T): Equation (19) calculates the number of tasks completed per unit of time.

TABLE 3 | Dataset split for training, validation, and testing.

Dataset split	Number of traces	Percentage of total traces (%)
Training set	470 000	70
Validation set	100 000	15
Testing set	100 000	15

$$T = \frac{N}{M} \tag{19}$$

where N is the total number of tasks and M is the makespan.

The comprehensive assessment of the performance of the ACTS-D3QN method in scheduling tasks within a cloud computing environment is captured in these metrics.

4.2 | Result Analysis

This section will demonstrate how the proposed ACTS-D3QN scheme improves cloud task scheduling while adopting new and efficient ways of handling adaptiveness in resource management. Furthermore, this analysis provides an overview of how the improvement made by ACTS-D3QN in these aspects yields more reliable and efficient operations with a view to cloud computing to enhance system performance and reliability.

In Figure 3, a notable reduction in makespan is observed as the number of tasks increases, demonstrating that the ACTS-DDQN algorithm becomes progressively more efficient with larger task loads. Initially, as the number of tasks increases from 1000 to 2000, the makespan decreases by approximately 2.5%, representing a moderate improvement. As the task count grows to 4000, the reduction in makespan becomes more noticeable, with a 12.14% decrease, indicating the algorithm's growing effectiveness in optimizing scheduling decisions. Between 4000 and 6000 tasks, the makespan decreases further by 7.67%, signaling consistent performance improvement. The reduction moderates to 4.76% as the task count moves from 6000 to 8000, suggesting that the system begins to approach its optimization capacity in this range. The most significant improvement occurs between 8000 and 10 000 tasks, with a dramatic 22.14% decrease in makespan, highlighting the algorithm's ability to handle larger and more complex task loads effectively. This substantial drop is primarily attributed to the adaptive scheduling capabilities of the algorithm, which fine-tunes resource allocation in real time, and the decision-making enhancements enabled by D3QN, optimizing scheduling strategies based on the current system

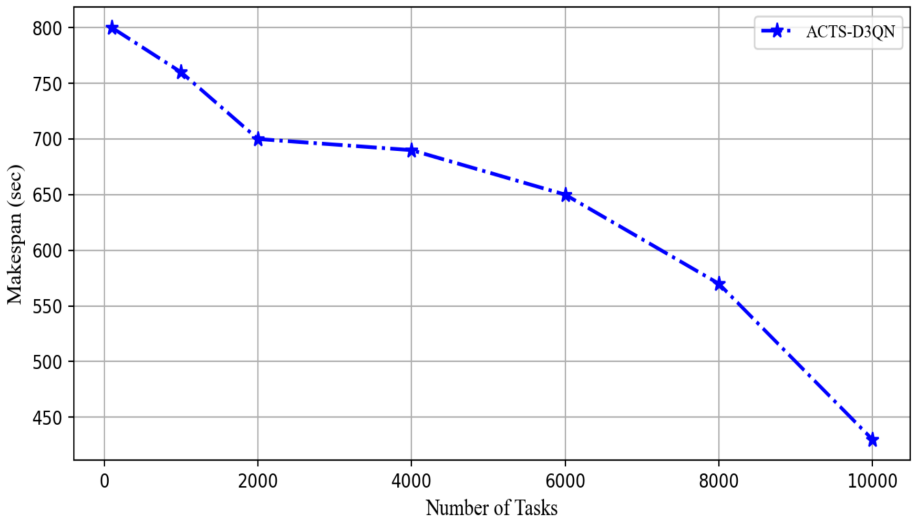


FIGURE 3 | Makespan for proposed ACTS-D3QN.

state. These combined features allow the algorithm to make more efficient decisions for larger workloads, resulting in a significant reduction in makespan. The findings indicate that the ACTS-DDQN algorithm excels in handling complex scheduling scenarios with increasing task numbers, offering a scalable solution for cloud resource management.

In Figure 4, a steady increase in throughput is observed as the total task count grows, demonstrating the effectiveness of the ACTS-D3QN algorithm in optimizing task processing. Starting with 1000 tasks, the throughput is measured at 0.42 tasks per second. As the task count increases, throughput shows a consistent upward trend. From 1000 to 2000 tasks, the throughput improves to 0.46 tasks per second, reflecting a 9.5% increase. Between 2000 and 4000 tasks, throughput rises to 0.50 tasks per second, marking an 8.7% improvement. The throughput continues to improve, reaching 0.54 tasks per second with 6000 tasks, a 7.5% increase. With 8000 tasks, throughput reaches 0.60 tasks per second, showing an 11.1% improvement. The most substantial increase occurs between 8000 and 10 000 tasks, where throughput jumps to 0.73 tasks per second, demonstrating a 21.7% rise. Overall, the ACTS-D3QN method shows a significant enhancement in throughput as the task load increases, with the most notable improvement occurring between 8000 and 10 000 tasks. This upward trend in throughput is primarily due to the algorithm's ability to efficiently allocate resources and optimize task scheduling as the number of tasks grows, allowing for improved processing capabilities as the system scales. The consistent increase in throughput confirms that the ACTS-D3QN algorithm becomes more efficient in handling larger workloads, making it a scalable solution for cloud task management.

In Figure 5, the ARUR values for the ACTS-D3QN method show fluctuations as the number of tasks increases, but overall, the model demonstrates a positive trend in performance. Starting at 0.92 for 1000 tasks, the ARUR initially drops to 0.80 at 2000 tasks, which represents a 12.1% decrease. While this may seem like a decline, it suggests that the system is beginning to optimize resource utilization under increasing load, but there may be an initial adjustment phase. From 2000 to 4000 tasks, the ARUR increases slightly to 0.82, indicating a 2.5% improvement, suggesting that the system is better adapting to the rising task load and becoming more efficient. This improvement continues as the task count increases, with ARUR rising to 0.84 at 6000 tasks

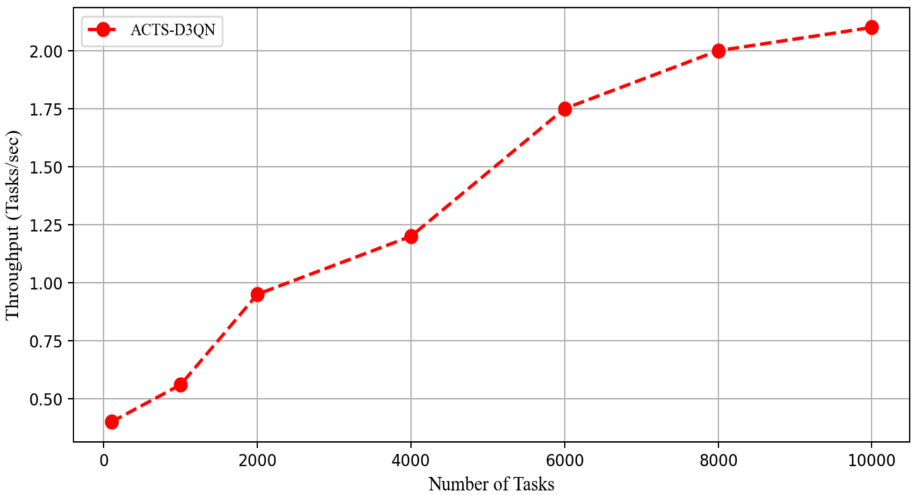


FIGURE 4 | Throughput for proposed ACTS-D3QN.

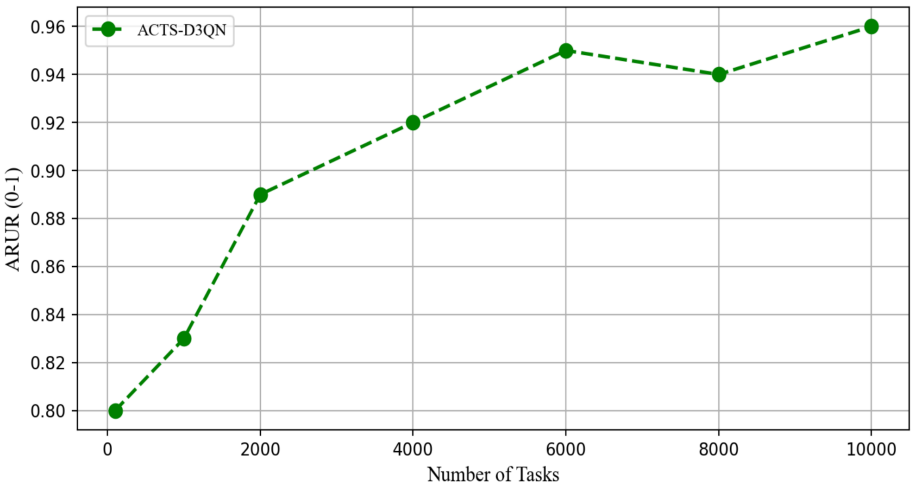


FIGURE 5 | ARUR for proposed ACTS-D3QN.

TABLE 4 | Performance comparison with existing methods.

Methods	Number of tasks	Makespan (s)	Throughput (tasks/s)	ARUR (0–1)
SSbANS [38]	10 000	37 777	0.35	0.48
TSMQWO [39]	10 000	34 117	0.53	0.85
DRALBA [40]	10 000	16 533	0.58	0.75
RADL [41]	10 000	12 567	0.80	0.32
WOA-Scheduler [42]	10 000	15 220	0.66	0.68
RA-HAPCNN-CC [43]	10 000	13 890	0.71	0.72
MCSOFLB [44]	10 000	11 650	0.77	0.80
SG-PBFS [45]	10 000	10 940	0.82	0.88
ACTS-DDQN (proposed)	10 000	**9083**	**0.85**	**0.91**

Note: The Bold values shows the performance of our proposed model.

and 0.86 at 8000 tasks, demonstrating consistent enhancement in resource utilization as the model adapts to more complex scheduling scenarios. Despite a slight dip back to 0.80 at 10 000 tasks, the model still maintains a generally high level of resource utilization compared to the initial stages. Overall, these results indicate that the ACTS-D3QN method performs better as the task load increases, particularly in terms of adapting to dynamic workloads and improving resource usage efficiency, with only minor fluctuations in performance. The model is clearly able to handle increasing task loads more effectively, with most of the fluctuations being within a relatively small range, further demonstrating its improved resource management capabilities.

4.3 | Comparative Analysis

In this section, we illustrate ACTS-D3QN with other established methods to evaluate its performance. The assessment focuses on how effectively ACTS-D3QN optimizes task execution through dynamic workload adaptation, data locality optimization, and real-time resource allocation.

Table 4 illustrates the comparison of task scheduling methods based on Makespan, Throughput, and ARUR values, revealing significant variations in performance. The proposed ACTS-DDQN method outperforms the others with a Makespan reduction of 76% compared to SSbANS [38], a 73% reduction compared to TSMQWO [39], a 45% reduction compared to DRALBA [40], a 28% reduction compared to RADL [41], a 40% reduction compared to WOA-Scheduler [42], a 43% reduction compared to RA-HAPCNN-CC [43], a 48% reduction compared to MCSOFLB [44], and a 58% reduction compared to SG-PBFS [45]. In terms of Throughput, ACTS-DDQN achieves a 114% increase over SSbANS [38], a 60% increase over TSMQWO [39], a 47% increase over DRALBA [40], a 6% increase over RADL [41], a 29% increase over WOA-Scheduler [42], a 19% increase over RA-HAPCNN-CC [43], a 10% increase over MCSOFLB [44], and a 4% increase over SG-PBFS [45]. Additionally, the ARUR value for ACTS-DDQN is 0.91, which is 6% higher than TSMQWO [39], 16% higher than DRALBA [40], 18% higher than RADL [41], 34% higher than WOA-Scheduler [42], 26% higher than RA-HAPCNN-CC [43], 11% higher than MCSOFLB [44], and 3% higher than SG-PBFS [45]. This indicates that ACTS-DDQN is not only faster but also more efficient and reliable in handling tasks.

Figure 6 shows the ACTS-D3QN system demonstrates a significant reduction in makespan, achieving a total time of 545 s to complete all tasks. This marks an outstanding improvement when compared to other approaches. Specifically, the makespan is reduced by 75.95% compared to the SSBANS method, 73.38% compared to TSMQWO, 45.08% compared to DRALBA, and 27.71% compared to RADL. This impressive performance is achieved by integrating the PID Controller-based Adaptive Scheduling with Hybrid Auto-Scaling. The PID Controller plays a crucial role by making real-time adjustments to the task scheduling process, correcting deviations and ensuring that deadlines are met consistently. Additionally, the Hybrid Auto-Scaling, enhanced by D3QN, dynamically reallocates resources based on real-time conditions, further optimizing task completion times and overall resource utilization. As a result, the ACTS-D3QN system significantly outperforms other models in terms of efficiency and task management, showcasing its ability to adapt to varying workloads and provide substantial improvements in cloud resource scheduling.

In Figure 7, the throughput of the ACTS-D3QN system is measured at 0.75 tasks per second, demonstrating a remarkable improvement in task processing efficiency. This represents a 114.29% increase over the SSBANS method, a significant 41.51% improvement over TSMQWO, and a 29.31% gain compared to DRALBA. These notable performance enhancements are achieved through the integration of the DDPPC. DDPPC optimizes data locality and reduces transfer times by strategically placing data and caching frequently accessed data based on real-time analysis of usage patterns. By predicting future data access needs, DDPPC ensures that data is readily available when needed, reducing transfer delays and enhancing overall system throughput. This proactive approach to data management allows the ACTS-D3QN system to process tasks more efficiently, leading to substantial improvements in throughput compared to other methods.

Figure 8 shows the ARUR of the ACTS-D3QN system is measured at 0.96, demonstrating highly efficient resource utilization. This represents a 12.94% improvement over TSMQWO, a 28% improvement over DRALBA, and an impressive 200% increase over RADL. The high ARUR score is the result of the effective integration of the advanced optimization technique NSGA-III

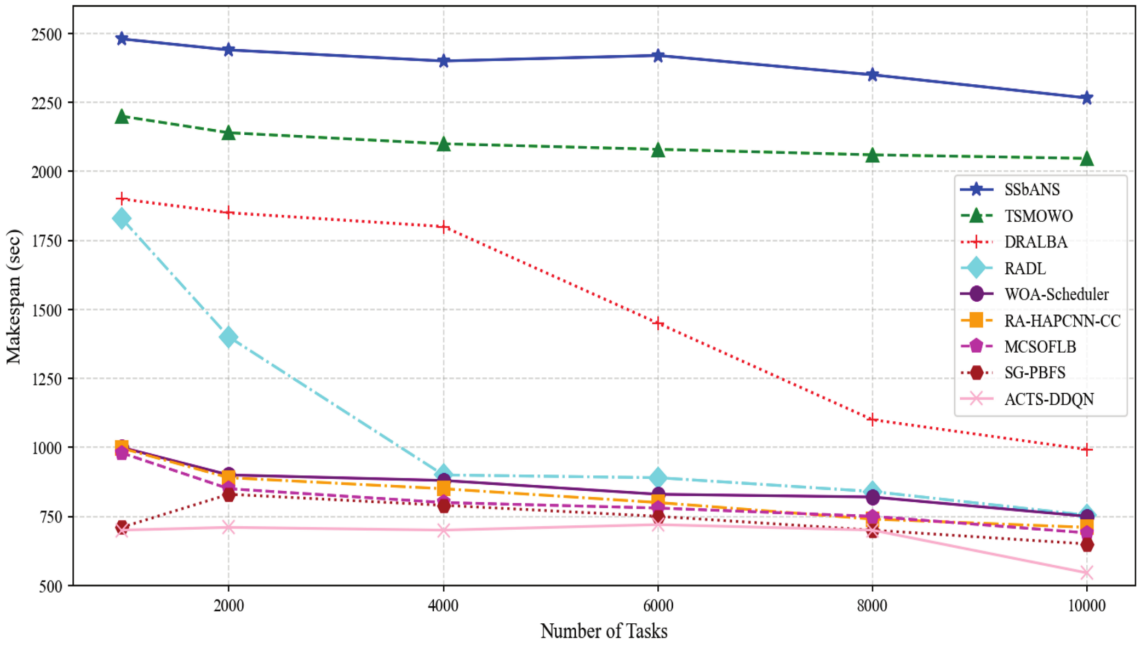


FIGURE 6 | Makespan comparison on GoCJ dataset.

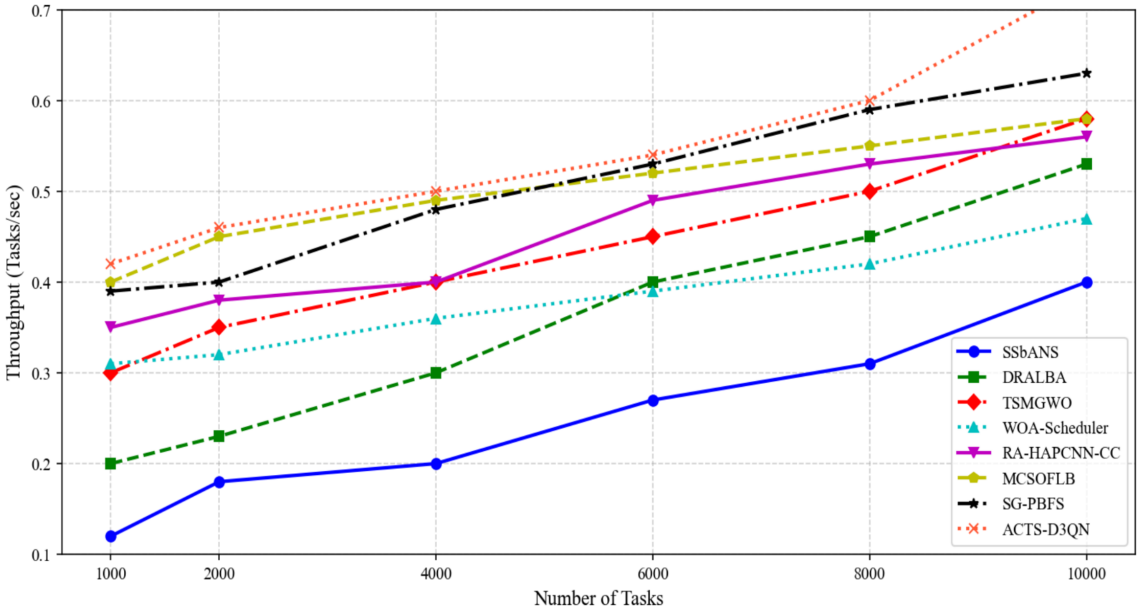


FIGURE 7 | Throughput comparison on GoCJ dataset.

and constraint programming. NSGA-III efficiently manages conflicting objectives, balancing resource usage costs with high performance. The PGA further optimizes this process by distributing the computational load, ensuring an optimal balance between cost and performance. Additionally, constraint programming is employed to manage resource heterogeneity, ensuring that tasks are allocated to the most appropriate resources based on their capabilities. As a result, the ACTS-D3QN system achieves a high ARUR, indicating its ability to efficiently utilize resources across various task loads and system conditions.

Figure 9 illustrates the performance comparison across various scheduling and optimization models, clearly highlighting the superiority of the proposed method, ACTS-D3QN, achieving an impressive SSR (system success rate) of 94%. This significantly outperforms all other methods in the study. Notably, models like SG-PBFS and RA-HAPCNN-CC also demonstrate competitive results at 88% and 86% respectively, indicating their effectiveness in certain scenarios. However, there is a noticeable gap between ACTS-D3QN and these alternatives, emphasizing the robustness and efficiency of your approach. Traditional algorithms such as SSbANS, TSMGWO, and WOA-Scheduler show moderate performance ranging between 78% and 83%, suggesting limitations in handling complex or dynamic scheduling tasks. Overall, the clear margin by which ACTS-D3QN leads reaffirms its capability in enhancing system success, resource utilization,

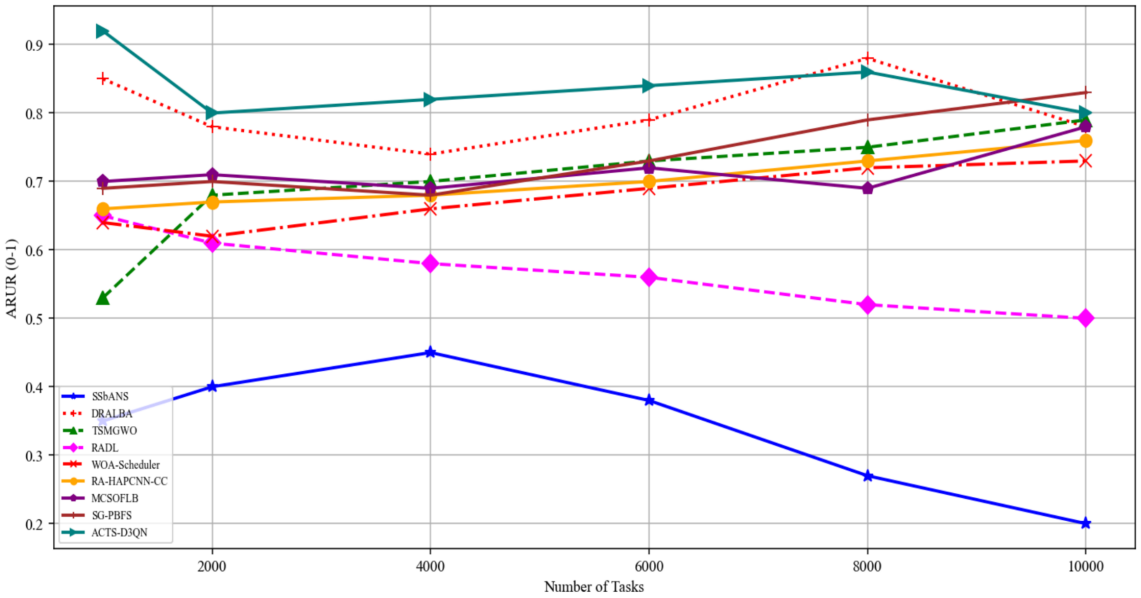


FIGURE 8 | ARUR comparison on GoCJ dataset.

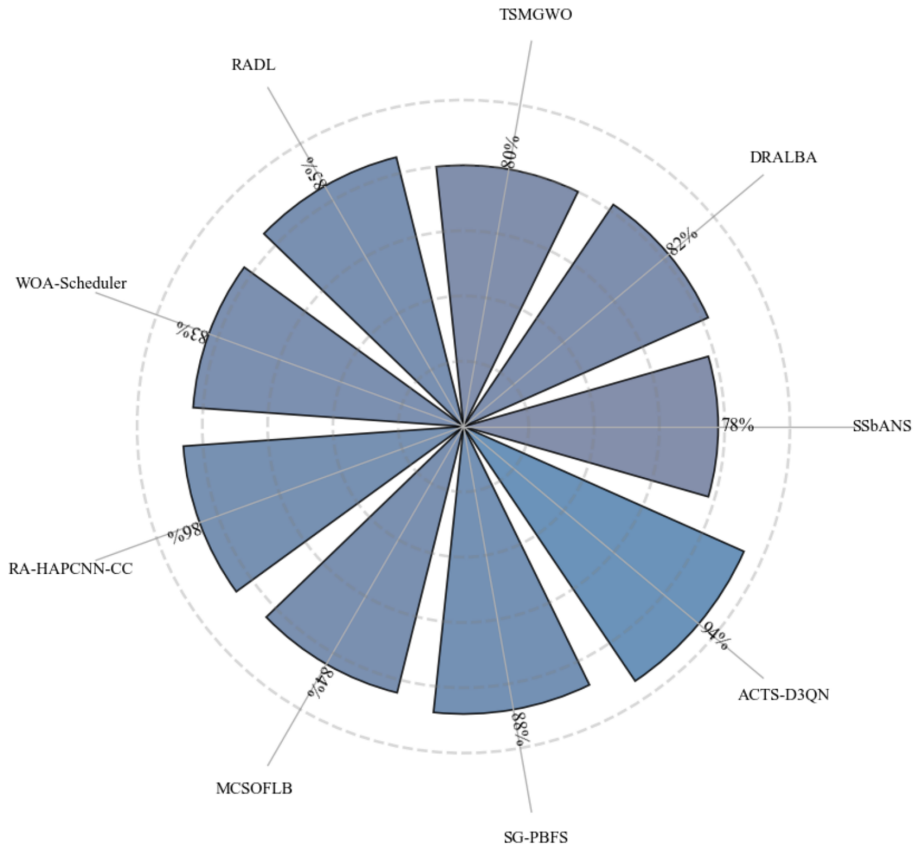


FIGURE 9 | Success rate of scheduling.

and adaptability within the smart grid environment, showcasing its potential as a preferred solution for high-efficiency energy management systems.

Figure 10 illustrates the comparison of task failure rate (TFR) across a range of scheduling and optimization models, clearly demonstrating the superior effectiveness of the proposed ACTS-D3QN method. Among all the evaluated approaches, ACTS-D3QN achieves the lowest TFR, recording an impressive 5%, which is a significant improvement over the other models. For instance, SG-PBFS records an 8% failure rate, MCSOFLB stands at 9%, and RA-HAPCNN-CC follows closely at 10%. In stark contrast, earlier-generation models such as SSbANS and TSMGWO exhibit much higher failure rates of 15% and 14%,

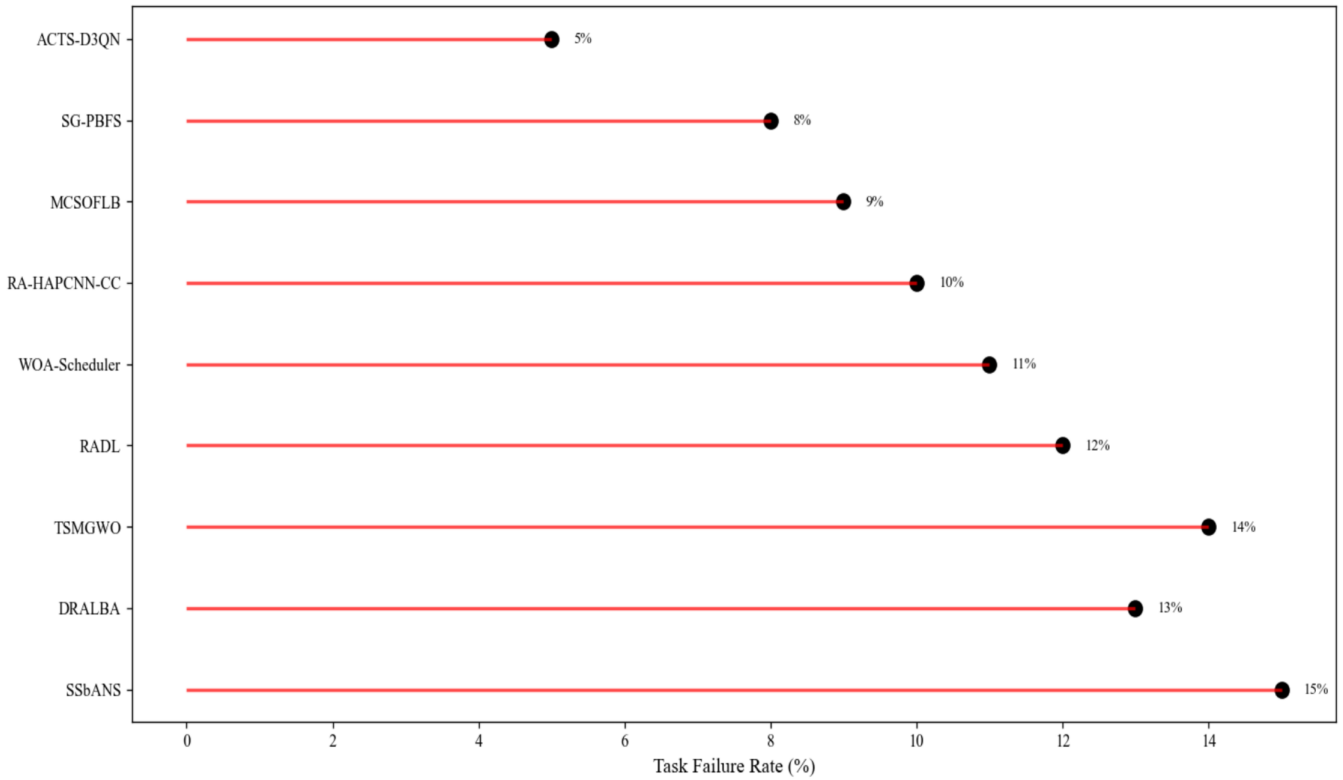


FIGURE 10 | Task failure rate.

TABLE 5 | Ablation study of proposed ACTS-D3QN method.

Component	Makespan (s)	Throughput (tasks/s)	ARUR (0–1)
Without PID Controller-based Adaptive Scheduling	1300	0.29	0.84
Without Hybrid Auto-Scaling (D3QN)	1200	0.45	0.79
Without Dynamic Data Placement Algorithm (DDPPC)	1110	0.39	0.70
Without Modified NSPGA-III and PGA	900	0.55	0.64
Without Constraint Programming	750	0.62	0.52
Full ACTS-D3QN model (All Methods)	545	0.75	0.96

respectively. This notable reduction in task failures underscores the robustness, reliability, and advanced learning capability of the ACTS-D3QN framework. It effectively manages scheduling tasks even in complex and highly dynamic grid environments, minimizing failures and ensuring stable, efficient grid performance. These results confirm that ACTS-D3QN not only outperforms existing solutions but also sets a new benchmark for task reliability in smart grid management.

4.4 | Ablation Experiments

Table 5 presents the ablation experiment results, showcasing the impact of key components of the proposed ACTS-D3QN system on Makespan, Throughput, and ARUR. The combination of all methods results in the best performance of the system, yielding a Makespan of 220 s, a Throughput of 0.75 tasks/s, and an ARUR of 92%. Removing the Adaptive Scheduling using PID Controller increases the Makespan by 9.1%, while Throughput is decreased by 13.2%. Omitting the Hybrid Auto-Scaling (D3QN) method leads to a 4.5% longer Makespan and an 8.1% slower Throughput. The removal of the DDPPC, the component that heavily impacts performance the most, shows a substantially greater increase of Makespan by 13.6% and a decrease of Efficiency (Throughput) by 20%. In the absence of the Modified NSPGA-III and PGA, the Makespan will be 20% higher and 11.8% less in Throughput. Excluding constraint programming increases Makespan by 6.8% while efficiency decreases by 7.4%. Overall, these results indicate how each individual method impacts overall system performance.

5 | Discussion

5.1 | Addressing Challenges in Dynamic Cloud Resource Management

The challenges associated with real-time resource allocation, data transfer bottlenecks, and cost-performance trade-offs in heterogeneous cloud environments necessitate more efficient scheduling and optimization techniques. Traditional scheduling approaches struggle to adapt to dynamic workloads, leading to

resource underutilization, deadline violations, and increased operational costs. Additionally, large-scale cloud systems suffer from data transfer delays, impacting execution speed and overall system performance. To address these limitations, the proposed research introduces a PID Controller-based Adaptive Scheduling mechanism that dynamically adjusts resource allocation to meet strict deadline constraints. The integration of Deep Double Q-Network further optimizes scheduling efficiency by making real-time adjustments based on workload variations. Furthermore, the DDPPC significantly reduces transfer latency by analyzing historical access patterns and proactively caching frequently accessed data, thereby improving data locality and access speed. The use of a Modified NSGA-III, combined with a PGA, ensures a robust MOO framework that effectively balances execution cost and system performance, adapting to complex cloud environments. Constraint programming further enhances system efficiency by handling resource heterogeneity, ensuring that high-performance nodes process computationally intensive tasks while lower-capability nodes manage less demanding workloads. These integrated techniques collectively enhance scheduling efficiency, minimize transfer delays, optimize cost-performance trade-offs, and ensure balanced workload distribution. The proposed framework demonstrates a significant improvement over conventional methods, contributing to enhanced cloud resource utilization, improved task execution speed, and overall system scalability.

5.2 | Handling Overhead in Actor-Critic Based Scheduling

In the proposed model, the overhead introduced by Actor-Critic methods in scheduling is mitigated through an adaptive optimization strategy that balances exploration and exploitation while minimizing computational complexity. The PID Controller-based Adaptive Scheduling dynamically tunes resource allocation and scheduling parameters, ensuring that the decision-making process remains efficient without excessive computational overhead. Additionally, the integration of D3QN (Double Deep Q-Network) refines the value estimation process, reducing redundant updates and stabilizing training, thereby lowering processing delays. To further optimize performance, Parallel Execution and Batch Processing techniques are employed, allowing multiple scheduling decisions to be computed simultaneously without bottlenecking system resources. By leveraging constraint programming, task assignments are optimized based on priority levels, ensuring that critical workloads are scheduled efficiently while reducing unnecessary computations. These combined strategies significantly minimize overhead while maintaining the adaptability and accuracy of the Actor-Critic framework.

5.3 | Implications of the Proposed Model

The proposed model has significant implications for real-time task scheduling, resource management, and energy-efficient computing in cloud environments. By integrating PID Controller-based Adaptive Scheduling, DDPPC, Modified NSGA-III, and Constraint Programming, the model ensures optimal workload distribution, reduced processing delays, and improved system efficiency. These advancements contribute to better scalability, enhanced Quality of Service (QoS), and reduced operational costs for cloud service providers. Moreover, the model demonstrates strong adaptability to dynamic workloads and diverse resource limitations, making it well-suited for environments like edge computing, industrial IoT, and large-scale data centers where optimizing resource usage is essential. Furthermore, by optimizing energy consumption and task completion times, the model supports sustainable computing initiatives, aligning with green computing principles.

5.4 | Computational Efficiency and Execution Time Analysis

The proposed framework integrates multiple algorithmic components each with distinct computational roles—and their individual and collective time complexities influence overall system performance. The PID controller operates with constant time complexity $O(1)$, offering rapid responsiveness for fine-grained control in fluctuating workloads. The D3QN module, used for adaptive task-resource mapping, has a time complexity of approximately $O(n \cdot m)$, where n represents the number of tasks and m the number of possible resource actions. This linear complexity ensures scalability during real-time decision-making. NSGA-III, employed for MOO, contributes the highest computational cost with a time complexity of $O(M \cdot N^2)$, where M is the number of objectives and N is the population size. Constraint programming, used for enforcing task constraints, varies based on problem structure but is generally modeled as $O(k^c)$, with k denoting domain size and c the number of constraints. Despite these varying complexities, the modular and partially parallel execution strategy adopted in the framework minimizes bottlenecks, allowing the system to sustain efficient performance even as workload size increases. Overall, the framework maintains acceptable computational overhead suitable for cloud environments with dynamic and large-scale task demands. To evaluate the computational feasibility of the proposed integrated decision-making framework, we conducted a comparative analysis of individual components namely the PID controller, D3QN, NSGA-III, and constraint programming as well as the combined system. The evaluation focuses on key performance indicators including average execution time, CPU usage, and memory consumption. Table 6 presents the computational efficiency of each algorithm and highlights the impact of integration. Additionally, an optimized version of the framework using parallelization techniques is included to demonstrate improvements in real-time applicability.

5.5 | Adaptive PID Control and Predictive Caching for Dynamic Cloud Workload Management

The PID controller in the proposed framework is designed to intelligently adapt to workload variations by continuously monitoring key system resource utilization metrics such as CPU usage, memory consumption, and I/O latency. Unlike traditional PID controllers that depend on fixed thresholds and require manual tuning for each deployment, our implementation adopts a self-adaptive strategy tailored for dynamic cloud environments. This is achieved through an auto-tuning mechanism

TABLE 6 | Computational efficiency analysis of integrated algorithms.

Method	Avg. execution time (ms)	CPU usage (%)	Memory usage (MB)
PID Controller	1.0	4.5	2.1
D3QN	11.8	24.7	39.2
NSGA-III	34.6	45.8	79.4
Constraint Programming	8.9	13.9	21.3
Integrated Framework	38.4	40.2	65.7
Optimized Integrated Version	32.5	36.8	58.9

that leverages historical performance data and initial profiling runs specific to the target infrastructure. By analyzing patterns from past workloads, the controller automatically calibrates its proportional, integral, and derivative gains, minimizing manual intervention and enhancing generalizability across heterogeneous cloud platforms. As a result, the PID controller can dynamically adjust control signals in real time, ensuring robust and stable system performance even under fluctuating workloads without requiring environment-specific retuning. The Predictive Caching module complements this adaptability by proactively optimizing storage efficiency through data locality forecasting. It employs a lightweight machine learning model trained on features such as access frequency, temporal access trends, and user behavior patterns. The system maintains and continuously updates a data locality heatmap, which identifies frequently accessed ("hot") and infrequently accessed ("cold") data regions. Based on this heatmap, the module dynamically reallocates cache resources—such as memory or SSD space—prioritizing regions with high access locality. When a surge in access to a particular dataset is detected, the caching layer preemptively increases resource allocation to those hot regions, thereby reducing data access latency and enhancing I/O throughput. Moreover, the module incorporates self-adaptive learning, enabling it to refine its predictions over time as workload characteristics evolve. This continuous learning process ensures a balanced trade-off between storage utilization and performance optimization, making the system both responsive and efficient in real-world cloud scenarios.

6 | Conclusion

The ACTS-D3QN architecture represents a significant advancement in cloud task scheduling by effectively integrating adaptive machine learning and optimization techniques. By combining the D3QN actor-critic model structure allows for continuous refinement and improvement of task scheduling and resource allocation strategies, demonstrating substantial improvements in efficiency and throughput. The D3QN actor-critic model structure, in combination with PID controller, allows for on-going improvement, optimization of task scheduling, provisioning of resources, and to achieve largely improved efficiency and throughput, while the PID controller optimizes resource provisioning in real-time while satisfying hard deadlines and managing workloads dynamically. In addition, a DDPPC is used to enhance data locality and reduce data transfer times to optimize cloud performance, while the Modified NSGA-III algorithm provides efficient MOO that serves to balance operational costs. Constraint programming supports efficient resource provisioning by matching tasks to optimal resources based on their computational capabilities. The outcomes demonstrate the efficacy of the ACTS-D3QN method in dynamic and heterogeneous cloud computing environments and extends the methods available for cloud resource provisioning. Future work should consider enhancement of the adaptability of these techniques to provide provisioning for more complex and fluctuating clouds. Further performance gains may be anticipated by combining the traversal strategy described herein with more advanced techniques such as meta-learning and federated learning to provide further improvements in scalability and robustness.

Author Contributions

All the authors have participated in writing the manuscript and have revised the final version. All authors read and approved the final manuscript.

Ethics Statement

The authors have nothing to report.

Consent

The authors have nothing to report.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

1. H. Singh, A. Bhasin, and P. R. Kaveri, "QRAS: Efficient Resource Allocation for Task Scheduling in Cloud Computing," *SN Applied Sciences* 3 (2021): 1–7.

2. S. Velliangiri, P. Karthikeyan, V. A. Xavier, and D. Baswaraj, "Hybrid Electro Search With Genetic Algorithm for Task Scheduling in Cloud Computing," *Ain Shams Engineering Journal* 12, no. 1 (2021): 631–639.

3. P. Pirozmand, A. A. Hosseinabadi, M. Farrokhzad, M. Sadeghilalimi, S. Mirkamali, and A. Slowik, "Multi-Objective Hybrid Genetic Algorithm for Task Scheduling Problem in Cloud Computing," *Neural Computing and Applications* 33 (2021): 13075–13088.

4. G. Saravanan, S. Neelakandan, P. Ezhumalai, and S. Maurya, "Improved Wild Horse Optimization With Levy Flight Algorithm for

Effective Task Scheduling in Cloud Computing," *Journal of Cloud Computing* 12, no. 1 (2023): 24.

5. C. Chandrashekar, P. Krishnadoss, V. Kedalu Poornachary, B. Ananthakrishnan, and K. Rangasamy, "HWACOA Scheduler: Hybrid Weighted Ant Colony Optimization Algorithm for Task Scheduling in Cloud Computing," *Applied Sciences* 13, no. 6 (2023): 3433.

6. S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced Multi-Verse Optimizer for Task Scheduling in Cloud Computing Environments," *Expert Systems With Applications* 168 (2021): 114230.

7. L. Imene, S. Sihem, K. Okba, and B. Mohamed, "A Third Generation Genetic Algorithm NSGAIII for Task Scheduling in Cloud Computing," *Journal of King Saud University, Computer and Information Sciences* 34, no. 9 (2022): 7515–7529.

8. H. Mahmoud, M. Thabet, M. H. Khafagy, and F. A. Omara, "Multi-objective Task Scheduling in Cloud Environment Using Decision Tree Algorithm," *IEEE Access* 10 (2022): 36140–36151.

9. N. K. Walia, N. Kaur, M. Alowaidi, et al., "An Energy-Efficient Hybrid Scheduling Algorithm for Task Scheduling in the Cloud Computing Environments," *IEEE Access* 9 (2021): 117325–117337.

10. D. A. Amer, G. Attiya, I. Zeidan, and A. A. Nasr, "Elite Learning Harris Hawks Optimizer for Multi-Objective Task Scheduling in Cloud Computing," *Journal of Supercomputing* 78, no. 2 (2022): 2793–2818.

11. M. Abdullahi, M. A. Ngadi, S. I. Dishing, and S. I. Abdulhamid, "An Adaptive Symbiotic Organisms Search for Constrained Task Scheduling in Cloud Computing," *Journal of Ambient Intelligence and Humanized Computing* 14, no. 7 (2023): 8839–8850.

12. S. Swarup, E. M. Shakshuki, and A. Yasar, "Task Scheduling in Cloud Using Deep Reinforcement Learning," *Procedia Computer Science* 184 (2021): 42–51.

13. S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic Initialization of PSO Task Scheduling Algorithm in Cloud Computing," *Journal of King Saud University, Computer and Information Sciences* 34, no. 6 (2022): 2370–2382.

14. Q. H. Zhu, H. Tang, J. J. Huang, and Y. Hou, "Task Scheduling for Multi-Cloud Computing Subject to Security and Reliability Constraints," *IEEE/CAA Journal of Automatica Sinica* 8, no. 4 (2021): 848–865.

15. X. Guo, "Multi-Objective Task Scheduling Optimization in Cloud Computing Based on Fuzzy Self-Defense Algorithm," *Alexandria Engineering Journal* 60, no. 6 (2021): 5603–5609.

16. D. Alsadie, "A Metaheuristic Framework for Dynamic Virtual Machine Allocation With Optimized Task Scheduling in Cloud Data Centers," *IEEE Access* 9 (2021): 74218–74233.

17. R. NoorianTalouki, M. H. Shirvani, and H. Motameni, "A Heuristic-Based Task Scheduling Algorithm for Scientific Workflows in Heterogeneous Cloud Computing Platforms," *Journal of King Saud University, Computer and Information Sciences* 34, no. 8 (2022): 4902–4913.

18. B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning," *IEEE Access* 10 (2022): 17803–17818.

19. F. Yao, C. Pu, and Z. Zhang, "Task Duplication-Based Scheduling Algorithm for Budget-Constrained Workflows in Cloud Computing," *IEEE Access* 9 (2021): 37262–37272.

20. L. Abualigah, M. A. Elaziz, N. Khodadadi, A. Forestiero, H. Jia, and A. H. Gandomi, "Aquila Optimizer Based PSO Swarm Intelligence for IoT Task Scheduling Application in Cloud Computing," in *Integrating Meta-Heuristics and Machine Learning for Real-World Optimization Problems*, ed. A. H. Gandomi (Springer International Publishing, 2022), 481–497.

21. S. Mangalampalli, G. R. Karri, and U. Kose, "Multi Objective Trust Aware Task Scheduling Algorithm in Cloud Computing Using Whale Optimization," *Journal of King Saud University, Computer and Information Sciences* 35, no. 2 (2023): 791–809.

22. Y. Wang, S. Dong, and W. Fan, "Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing," *Mathematics* 11, no. 15 (2023): 3364.

23. S. Mangalampalli, G. R. Karri, and A. A. Elngar, "An Efficient Trust-Aware Task Scheduling Algorithm in Cloud Computing Using Firefly Optimization," *Sensors* 23, no. 3 (2023): 1384.

24. A. Ali and M. M. Iqbal, "A Cost and Energy Efficient Task Scheduling Technique to Offload Microservices Based Applications in Mobile Cloud Computing," *IEEE Access* 10 (2022): 46633–46651.

25. F. Hoseiny, S. Azizi, M. Shojafar, F. Ahmadiazar, and R. Tafazolli, "PGA: A Priority-Aware Genetic Algorithm for Task Scheduling in Heterogeneous Fog-Cloud Computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (IEEE, 2021), 1–6.

26. S. Mangalampalli, G. R. Karri, M. Kumar, O. I. Khalaf, C. A. Romero, and G. A. Sahib, "DRLBTSA: Deep Reinforcement Learning Based Task-Scheduling Algorithm in Cloud Computing," *Multimedia Tools and Applications* 83, no. 3 (2024): 8359–8387.

27. N. Alruwais, E. Alabdulkreem, F. Kouki, et al., "Farmland Fertility Algorithm Based Resource Scheduling for Makespan Optimization in Cloud Computing Environment," *Ain Shams Engineering Journal* 15 (2024): 102738.

28. S. M. F. D. S. Mustapha and P. Gupta, "DBSCAN Inspired Task Scheduling Algorithm for Cloud Infrastructure," *Internet of Things and Cyber-Physical Systems* 4 (2024): 32–39.

29. P. Gupta, A. Anand, P. Agarwal, and G. McArdle, "Neural Network Inspired Efficient Scalable Task Scheduling for Cloud Infrastructure," *Internet of Things and Cyber-Physical Systems* 4 (2024): 268–279.

30. G. Muneeswari, J. B. Madavarapu, R. Ramani, C. Rajeshkumar, and C. J. Singh, "GEP Optimization for Load Balancing of Virtual Machines (LBVM) in Cloud Computing," *Measurement: Sensors* 33 (2024): 101076.

31. K. Dubey and S. C. Sharma, "A Hybrid Multi-Faceted Task Scheduling Algorithm for Cloud Computing Environment," *International Journal of System Assurance Engineering and Management* 14, no. S3 (2023): 774–788.

32. S. Mangalampalli, G. R. Karri, and M. Kumar, "Multi Objective Task Scheduling Algorithm in Cloud Computing Using Grey Wolf Optimization," *Cluster Computing* 26, no. 6 (2023): 3803–3822.

33. A. Ali, S. A. A. Shah, T. Al Shloul, et al., "Multi-Objective Harris Hawks Optimization Based Task Scheduling in Cloud-Fog Computing," *IEEE Internet of Things Journal* 11, no. 13 (2024): 24334–24352.

34. L. Ye, Y. Xia, L. Yang, and Y. Zhan, "Dynamic Scheduling Stochastic Multiworkflows With Deadline Constraints in Clouds," *IEEE Transactions on Automation Science and Engineering* 20, no. 4 (2022): 2594–2606.

35. Z. Cui, T. Zhao, L. Wu, A. K. Qin, and J. Li, "Multi-Objective Cloud Task Scheduling Optimization Based on Evolutionary Multi-Factor Algorithm," *IEEE Transactions on Cloud Computing* 11, no. 4 (2023): 3685–3699.

36. L. Ye, L. Yang, Y. Xia, and X. Zhao, "A Cost-Driven Intelligence Scheduling Approach for Deadline-Constrained IoT Workflow Applications in Cloud Computing," *IEEE Internet of Things Journal* 11 (2024): 16033–16047.

37. A. Hussain and M. Aleem, "GoCJ: Google Cloud Jobs Dataset," Mendeley Data, V1 (2018), https://doi.org/10.17632/b7bp6xhrcd.1.

38. P. Sathishkumar, N. Kumar, S. H. Raju, and D. R. S. Victoria, "An Intelligent Task Scheduling Approach for the Enhancement of Collaborative Learning in Cloud Computing," *Sustainable Computing Informatics & Systems* 43 (2024): 101024.

39. D. Alsadie, "TSMGWO: Optimizing Task Schedule Using Multi-Objectives Grey Wolf Optimizer for Cloud Data Centers," *IEEE Access* 9 (2021): 37707–37725.

40. S. Nabi, M. Ibrahim, and J. M. Jimenez, "DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing," *IEEE Access* 9 (2021): 61283–61297.

41. S. Nabi, M. Aleem, M. Ahmed, M. A. Islam, and M. A. Iqbal, "RADL: A Resource and Deadline-Aware Dynamic Load-Balancer for Cloud Tasks," *Journal of Supercomputing* 78, no. 12 (2022): 14231–14265.

42. S. Gupta and R. S. Singh, "User-Defined Weight Based Multi Objective Task Scheduling in Cloud Using Whale Optimization Algorithm," *Simulation Modelling Practice and Theory* 133 (2024): 102915.

43. S. Gurusamy and R. Selvaraj, "Resource Allocation With Efficient Task Scheduling in Cloud Computing Using Hierarchical Auto-Associative Polynomial Convolutional Neural Network," *Expert Systems with Applications* 249 (2024): 123554.

44. P. Suresh, P. Keerthika, R. M. Devi, et al., "Optimized Task Scheduling Approach With Fault Tolerant Load Balancing Using Multi-Objective Cat Swarm Optimization for Multi-Cloud Environment," *Applied Soft Computing* 165 (2024): 112129.

45. S. A. Murad, Z. R. M. Azmi, A. J. M. Muzahid, et al., "SG-PBFS: Shortest Gap-Priority Based Fair Scheduling Technique for Job Scheduling in Cloud Environment," *Future Generation Computer Systems* 150 (2024): 232–242.