Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Research on Cloud Task Scheduling Problem Based on Pareto Optimization Strategy for Deep Reinforcement Learning

Zhou Wenqin
Tianshui Normal University
Dai Lu
1973823800@qq.com
Tianshui Normal University
Wu Zhoulong
Tianshui Normal University
Hua Na
Tianshui Normal University

Research Article

Keywords: Cloud computing, Task scheduling, Deep reinforcement learning, Pareto optimization, Multi-objective optimization

Posted Date: November 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11419144/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Research on Cloud Task Scheduling Problem Based on Pareto Optimization Strategy for Deep Reinforcement Learning

Zhou Wenqin[1], Dai Lu[2], Wu Zhoulong[2], Hua Na[2]

(School of Electronic Information and Electrical Engineering, Tianshui Normal University, Tianshui 741001, Gansu, China)

Abstract: Cloud data centers face highly dynamic cluster loads and sharply heterogeneous task demands, which place heavy pressure on task scheduling. Most existing schedulers rely on local state features, adapt poorly to load fluctuations, and struggle to balance competing objectives. This paper proposes ITS-D3QN (Interactive Two-Stage D3QN), a deep reinforcement learning scheduler built on D3QN (Dueling Double Deep Q-Network) and driven by a Pareto optimization strategy. The method has three components. First, a two-layer scheduling model is formulated in which the lower layer optimizes task completion time, system energy consumption, and resource utilization. Second, the upper layer uses the Pareto hypervolume (HV) to capture cluster-wide load and resource conditions, and the scheduling problem is cast as a Markov decision process. Third, an incremental sampling mechanism is introduced to improve convergence and adaptability to non-stationary load. Simulations on the Alibaba Cluster Trace v2018 show that, relative to the compared algorithms, ITS-D3QN reduces average task response time by approximately 4.75% and average energy consumption per task by approximately 4.32%, while raising average resource utilization by approximately 5.62%. These results indicate that the method achieves balanced scheduling in complex cloud environments.

Keywords: Cloud computing; Task scheduling; Deep reinforcement learning; Pareto optimization; Multi-objective optimization

0 Introduction

Cloud computing and data center technologies have developed rapidly, and large numbers of terminal services and computationally intensive, heterogeneous tasks now arrive concurrently [1-2]. This places strict demands on the computing power and resource allocation of data centers, and the scheduling policy directly determines the overall operational efficiency of the system. Large-scale cloud tasks typically arrive at random and differ widely in their resource requirements, continuously consuming core hardware resources such as CPU and memory. Because resources are limited while task demands vary sharply, scheduling decisions based only on local state tend to cause resource hoarding, node overload, or idle capacity [3]. One remedy is to constrain scheduling actions that violate resource constraints and increase system loss [4]. Deep reinforcement learning (DRL) offers a further route: an agent can learn a stable and efficient policy from interaction and thereby optimize several objectives at once.

Multi-objective scheduling based on DRL has been studied extensively. The dominant approach is decomposition and coordination: the objectives are first decomposed into a set of scalar sub-problems by linear weighting, Chebyshev scalarization, or MOEA/D, and each sub-problem is assigned to a lightweight agent

Zhou Wenqin

zhouwenqin@tsnu.edu.cn

Dai Lu

1973823800@qq.com

Wu Zhoulong

2309785001@qq.com

Hua Na

3358206724@qq.com

[1] Associate Professor of Tianshui Normal University, Mainly engaged in research on next-generation internet technologies and cloud computing.

[2] Tianshui Normal University School of Electronic Information and Electrical Engineering, Institute of Artificial Intelligence Education,741001Tianshui.

trained in parallel [5]; the sub-problems are then synchronized through neighborhood parameter migration, a shared experience pool, or a centralized global network, producing schedulers that trade off latency, energy consumption, load balancing, and quality of service [6].

Recent studies illustrate this line of work. Chen et al. [7] combined a maximum-entropy framework with graph convolutional networks and an adaptive loss function, improving adaptability under dynamic scheduling. Zhang et al. [8] addressed the instability of inference-time randomness under dynamic load with an online scheduling algorithm supervised by policy entropy; the method performs real-time entropy calibration without altering the training framework and jointly optimizes energy consumption and average response time. Sheng et al. [9] targeted insufficient multi-objective optimization at the edge with an improved competitive deep Q-network that suppresses value overestimation and improves sample utilization, coordinating response time and energy consumption. Naaz et al. [10] proposed a lightweight framework that couples a single-layer LSTM with DQN and uses a reward function combining delay, throughput, and reliability to balance QoS. Azari and Jalili [11] applied DRL to dynamic computation offloading and resource allocation, coordinating value and policy networks to reduce delay and improve resource utilization. Ye and Shen [12] addressed Spark cluster scheduling with an action-branching DRL method that uses graph embedding to capture the structure of workflows, shrinking the action space and improving efficiency. Xu et al. [13] developed a cloud data center scheduler based on adaptive state optimization and dynamic step size, reducing latency and energy consumption through high-dimensional state optimization and a JS-divergence strategy. Chraibi et al. [14] formulated dynamic multi-objective workflow scheduling as a double-layer deep Q-network with prioritized experience replay, improving multi-objective performance under complex task dependencies through fixed network dimensions and random-subset training.

The Pareto set, that is, the non-dominated solutions of a multi-objective problem, can be summarized by the hypervolume (HV), a widely used indicator of multi-objective quality [15-16]. Most existing DRL schedulers, however, have limited state representations [17] and extract little cluster-level information, so the agent perceives the environment incompletely and performs poorly under complex load patterns [18]. This paper therefore proposes a DRL scheduler with macro-state perception that uses HV to adapt the weights assigned to different objectives [19], targeting task response time, server energy consumption, and resource utilization.

1 Cloud Computing Task Scheduling Model

1.1 System model

A cloud data center receives a large number of tasks with diverse computing requirements. The submitted tasks are denoted $Task = \{T1, T2, \dots, Ti\}$, the servers that host them are denoted $S = \{S_1, S_2, \dots, S_l\}$, and the virtual machines on which the tasks execute are denoted $VM = \{vm_1, vm_2, \dots, vm_j\}$. Each virtual machine operates independently. Figure 1 illustrates the scheduling model. Submitted tasks first enter the global task queue Q, where they are buffered and preprocessed [20]. A DRL agent then makes the matching decision and dispatches eligible tasks to the waiting queue of the selected virtual machine, and resources are allocated on a first-come, first-served basis [21].

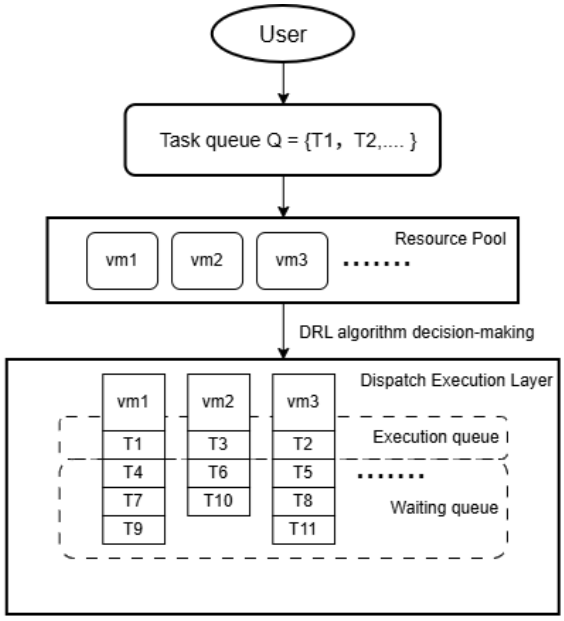


Figure 1 Cloud computing resource scheduling framework

1.2 Problem Definition

This paper optimizes three objectives: task completion time, system energy consumption, and resource utilization. The tasks submitted by users have the following attributes:

$$T_i = \langle cpu_i, mem_i, t_i \rangle \tag{1}$$

Here, cpu_i represents the CPU requirement of the task, mem_i represents the memory requirement of the task, and t_i represents the execution time of the task.

The virtual machines hosted on the servers are denoted as

$$vm_j^l = \langle cpu_j^l, mem_j^l, P_{\text{sj}}^l, P_{\text{mj}}^l \rangle \tag{2}$$

where cpu_j^l represents the CPU capacity of the virtual machine, mem_j^l represents the memory capacity of the virtual machine, P_{sj}^l represents the static power of the virtual machine, and P_{mj}^l represents the full-load power of the virtual machine.

The servers in the system can load different numbers of virtual machines. Suppose the system assigns task t_1 to virtual machine vm_1. When task t_2 is assigned or enters the waiting queue Q_w of vm_1, the completion time of the task is the sum of the execution time of t_1 and t_2; or it is assigned to an idle virtual machine, in which case the completion time of the task is the execution time of t_2. If all the virtual machines are occupied, the target virtual machine with the highest priority will be selected for deployment. If none of them meet the requirements, task t_2 will be discarded; additionally, if task t_2 waits in Q_w for a certain period of time, it will also be discarded [22-24].

The waiting time of a task is therefore

$$T_{w_{ij}}^l = \sum_{m=1}^{M} T_{e_{ij}}^l \tag{3}$$

Here, M represents the number of tasks in the waiting queue of vm_j^l, $T_{e_{ij}}^l$ is the execution time of the tasks in vm_j^l, and the task completion time $T_{r_{ij}}^l$ is expressed as:

$$T_{r_{ij}}^l = T_{e_{ij}}^l + T_{w_{ij}}^l \tag{4}$$

Resource utilization combines CPU and memory utilization:

$$\begin{cases} U^{\text{cpu}}(t) = \sum_{j=1}^{N} \frac{CPU_{\text{used}j}^t}{cpu_j^l}, U^{\text{mem}}(t) = \sum_{j=1}^{N} \frac{MEM_{\text{used}j}^t}{mem_j^l} \\ \bar{U}^{\text{cpu}} = \frac{1}{T}\sum_{t=1}^{T} U^{\text{cpu}}(t), \bar{U}^{\text{mem}} = \frac{1}{T}\sum_{t=1}^{T} U^{\text{mem}}(t) \\ \bar{U}_{\text{ep}} = 0.7 \times \bar{U}^{\text{cpu}} + 0.3 \times \bar{U}^{\text{mem}} \end{cases} \tag{5}$$

where N represents the number of virtual machines. At step t $(t = 1, \dots, T)$, $CPU_{\text{used}j}^t$ and $MEM_{\text{used}j}^t$ represent the CPU and memory usage of the virtual machine, while $U^{\text{cpu}}(t)$ and $U^{\text{mem}}(t)$ represent the utilization rates of all virtual machines. Subsequently, the weighted average of the average CPU utilization rate $\bar{U}^{\text{cpu}}$ and the average memory utilization rate $\bar{U}^{\text{mem}}$ is obtained to calculate the average resource utilization rate $\bar{U}_{\text{ep}}$ for one round.

Energy consumption of a virtual machine is nonlinear: dynamic power depends on CPU utilization [25], and the average energy per task is

$$E_{\text{task}}(t) = P_{\text{sj}}^l + P_{\text{mj}}^l \times \frac{CPU_{\text{used}j}^t}{cpu_j^l} \tag{6}$$

Weights are then assigned to response time, resource utilization, and per-task energy consumption, so that the three objectives are optimized jointly through trade-off.

Following [15], the Pareto solution set $Y = \{y_1, \dots, y_r\}$ is defined, where $y_r = \langle T_{r_{ij}}^l, \bar{U}_{\text{ep}}, E_{\text{task}} \rangle$. When the hypervolume of Y is maximized, the set converges to the Pareto optimum. Exact computation of the union of three-dimensional hypervolumes is expensive, so this paper estimates HV by Monte Carlo sampling [26]: points $s^{(1)}, \dots, s^{(N)}$ are drawn uniformly from the fixed region $[0,1]^d$, and the hypervolume is approximated by the fraction of samples covered by the front, that is

$$HV = \frac{1}{N}\sum_{k=1}^{N} 1\left[\exists Y \in F : s^{(k)} \preceq Y\right] \tag{7}$$

Here, F denotes the non-dominated solutions of Y, that is, the Pareto front. HV therefore quantifies how well the front covers the objective space within a sampling batch. At the n-th sampling, let F_{before} and F_{after} denote the fronts before and after the update; the hypervolume increment is then

$$\Delta HV = \frac{1}{n}\sum_{n=1}^{n^*} [\mathrm{HV}(F_{\mathrm{after}}) - \mathrm{HV}(F_{\mathrm{before}})] \quad (8)$$

Quantifying the expansion of the dominated region in each iteration, and iterating this estimate externally, drives further improvement of all three objectives.

2 Design of the ITS-D3QN Algorithm

2.1 Algorithm framework and network structure

ITS-D3QN adopts a two-layer architecture (Figure 2): an upper management layer operating on a macro time scale, and a lower execution layer operating on the real-time task scale. The upper layer takes the long-term load trend of the cluster as input, uses HV as the core reward signal, and outputs a multi-objective preference weight vector that re-parameterizes the lower layer, driving the solution toward the global Pareto front. The lower layer handles single-step placement: it receives the weight instruction, the real-time task stream, and the load state from the upper layer, and obtains physical performance indicators from the scheduler, thereby closing the training loop.

Within a task step, the upper layer selects one weight vector as the components of the lower layer's multi-objective reward, while the solutions produced by the lower layer enter a sliding window that serves as the upper layer's input and yields the HV increment. Training samples are then drawn from the lower layer's replay buffer, forming a loop with environmental feedback.

Both layers retain the D3QN network architecture, which consists of the following components.

1) Dueling network: To enhance the sample utilization rate, the dueling network divides the Q function into the state value $V_\theta(s)$ and the advantage function $A_\theta(s,a)$. Additionally, while the network is learning the states and actions separately, a term for the mean removal of the advantage value is added to ensure the convergence of the neural network. The definition is as shown in Equation (17):

$$Q_\theta(s,a) = V_\theta(s) + A_\theta(s,a) - \frac{1}{|A|}\sum_{a'\in A} A_\theta(s,a') \quad (9)$$

2) Double Q network: To mitigate the overestimation of Q-values caused by directly using the target network in ordinary DQN, the double Q network splits it into two steps. Firstly, it selects actions from the online network θ. Additionally, to reduce ineffective exploration, an action mask is added. For the next state s', on the feasible set $F(s')$, it is defined as:

$$Q_\theta(s',a) = \begin{cases} Q(s',a) & a \in F(s') \\ -10^9 & a \notin F(s') \end{cases} \quad (10)$$

which yields

$$a^* = \arg\max_{a\in A} Q_\theta(s',a) \quad (11)$$

where a^* represents the action with the highest Q value. To avoid the interference of noise, the value is further evaluated using the target network θ^-:

$$Q_{\mathrm{next}} = Q_{\theta^-}(s',a^*) \quad (12)$$

where Q_{next} represents the Q value of a^* in state s'. Both networks utilize separate dueling networks. At this point, the loss function $\mathcal{L}(\theta)$ is expressed as:

$$\begin{cases} z = r + (1-D)\cdot\gamma\cdot Q_{\mathrm{next}} \\ \mathcal{L}(\theta) = \frac{1}{B}\sum_{i=1}^{B} \ell_{\mathrm{Huber}}(Q_\theta(s_i,a_i) - z) \end{cases} \quad (13)$$

In the network state D, the TD target z is calculated. Then, the difference between the predicted value and z is calculated using the Huber loss, and the average of B samples is taken to obtain the total loss. After calculating $\mathcal{L}(\theta)$, parameter gradient clipping is performed for the parameters, and then the θ is updated through the Adam optimizer in the reverse direction; θ^- is the parameter copy of θ within the fixed task step to achieve update, reducing the oscillation caused by chasing the same network.

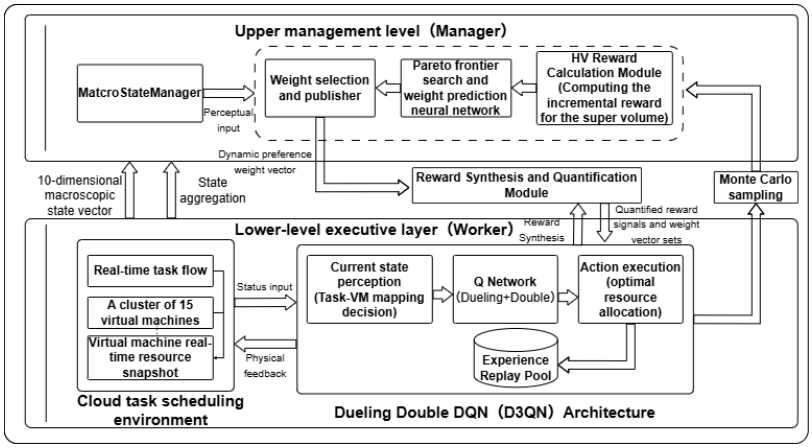


Figure 2 Two-layer interactive scheduling framework

2.2 Markov decision elements

As Figure 2 shows, scheduling in both layers is modeled as a Markov decision process with three elements: state space, action space, and reward function.

1) State space

The model is organized into an upper layer and a lower layer. The lower layer is the execution layer, whose state space is defined as

$$d_w = \left\{cpu_i, mem_i, t_i, t, N(U^{\text{cpu}}, U^{\text{mem}}, Q_w, T^l_{w_{ij}}), W\right\} \quad (14)$$

Here, W represents the set of multi-objective weight vectors. The execution layer adjusts the hyperparameters according to task requirements, virtual machine status, and the weights assigned by the upper layer, correcting the training direction in time. The upper layer is the management layer, and its state space dimension is expressed as:

$$d_m = \left\{cpu_i, mem_i, \bar{U}_{\text{ep}}, \sigma_U^2, P_{\text{over}}, W\right\} \quad (15)$$

All dimensions in Eq. (10) are obtained by averaging over the sampling window. σ_U^2 represents the variance of the average utilization rate. P_{over} is the timeout rate. The upper layer thus receives training information through feedback from the lower layer and adjusts the multi-objective weights.

2) Action Space

For the execution layer, tasks are assigned to virtual machines according to the state-action mapping; the action space is

$$a^w = \left\{vm_1, vm_2, \ldots, vm_j\right\} \quad (16)$$

The management layer offers the lower layer different combinations of multi-objective hyperparameters; its action space is

$$a^m = \{W_1, W_2, \ldots, W_n\} \quad (17)$$

3) Reward Function

Scheduling here is a multi-objective problem. Coupling the scheduling system with the scheduling process, the reward at the execution layer consists of an event term and a multi-objective shaping term, defined as

$$R^W = r_t + r_w \quad (18)$$

where r_t indicates whether task allocation succeeded or failed, and r_w is expressed as:

$$\begin{cases} W = \left(W_u, W_w, W_\sigma, W_q, W_p, W_e, W_{ms}\right) \\ \phi_t = \left(\bar{U}_{\text{ep}}, -T_t, -S_t, -Q_t, -O_t, (1-E_t), (1-M_t)\right) \\ r_w = W \times \phi_t \end{cases} \quad (19)$$

At step t, the weight reward function ϕ_t is composed of the normalized resource utilization $\bar{U}_{\text{ep}}$, the sum of the waiting times of tasks in the queue T_t, load balancing S_t, queue congestion degree Q_t, task timeout rate O_t, unit energy consumption E_t, and maximum completion time M_t. The weights W are assigned to the above seven items respectively.

The management layer does not update at every environment step. Instead, it computes a macro reward after a fixed window of steps and writes the result to the experience buffer.

The reward is defined as

$$R^M = \alpha_{\text{HV}} \cdot \Delta\text{HV} + r_{\text{i}} \quad (20)$$

where ΔHV represents the increment of the hypervolume, and α_{HV} is the amplification coefficient, which is used to match the gradient of the Q network and the network reward r_{i}, where:

$$\begin{cases} r_{\text{i}} = \eta \cdot (\bar{Q} - B) \\ B \leftarrow (1-\beta)B + \beta\bar{Q} \end{cases} \quad (21)$$

where η represents the amplification coefficient, $\bar{Q}$ is defined as the average of the maximum Q values of each action for each sample under the feasible VM mask, B is the relative exponential sliding baseline, β is the update step size of the exponential sliding average, which is the deviation of the sample mean of the lower-level Q values on the available actions relative to the exponential sliding average, and it characterizes the potential changes in the learning and execution of the lower-level.

2.3 Dynamic preference switching mechanism

Fixed weights cannot follow dynamic load, so this section introduces a dynamic preference switching mechanism driven by HV feedback. Weights are updated every 200 task steps within a window T_m. On the basis of the mean reward over that window and the combined selection of upper-layer rewards, the lower-layer reward is recombined so that the policy operates under the updated preference. Table 1 lists the weight configurations. Each configuration encodes a distinct scheduling preference, so changing the weights shifts the operating point along the Pareto front.

Table 1 Multi-objective weight configurations

Group number	W_u	W_w	W_σ	W_q	W_p	W_e	W_{ms}	Strategic positioning
0	1.6	3.0	1.2	1.5	4.0	1.8	2.0	Baseline (default)
1	4.2	1.0	1.0	1.0	1.0	1.1	1.0	Priority given to utilization rate
2	1.2	5.2	1.0	2.0	8.0	2.0	3.6	Priority for delays and overtimes
3	1.5	2.0	4.2	1.5	2.0	1.4	1.5	Load balancing priority
4	1.1	2.0	1.0	1.0	3.0	2.8	2.6	Energy conservation, time priority
5	2.8	2.2	1.2	1.4	2.5	2.0	2.2	Trade-off between utilization rate and energy efficiency
6	1.4	3.8	1.3	2.8	4.5	1.8	2.8	Congestion Control
7	2.2	2.4	2.6	1.2	2.0	1.6	1.8	Utilization balance
8	1.0	2.8	1.0	1.0	2.5	3.6	3.0	Energy conservation takes priority.
9	2.0	3.0	1.5	1.6	3.8	2.4	2.8	Robust comprehensive trade-off

2.4 Incremental sampling strategy

Uniform experience replay offers low sample utilization and may lose critical transitions, Monte Carlo estimation of HV is noisy, and preference switching is unstable. This section therefore introduces an incremental sampling strategy for multi-objective scheduling. The number of independent repeated samples at the upper layer, n^*, is increased sequentially and is defined as

$$n^* = min\left\{n \in \{R_{min}, \dots, R_{max}\}: z \cdot \frac{D_n}{n} \leq \varepsilon\right\} \quad (22)$$

Here, z is the confidence critical value. After the Pareto archive is updated in each macro window, the hypervolume increment is re-estimated by independent Monte Carlo runs: starting from R_{min}, n is increased step by step, and the procedure stops once the standard error of the mean falls below the accuracy threshold; otherwise $n^* = R_{max}$. This balances estimation stability against computational cost. The procedure is summarized in Algorithm 1.

Algorithm 1 ITS-D3QN Algorithm

Input: Virtual machine set VM, task sequence $Task$, weight set W.

Output: upper- and lower-layer policies; scheduling performance indicators.

1. Initialize the network; experience pool, Pareto pool A; task step g.
2. for training rounds $e = 1,2,...$do
3. Reset the environment; clear the window and counters
4. for Task Step $g = 1,2,...$do
5. Obtain the lower-level status and calculate the feasible VM
6. Perform the action by using the greedy strategy or DDQN to select a^*
7. Carry out the scheduling, calculate R^W and store $(s,a,r,s')^W$ in the experience pool
8. if $g\ mod\ 200 = 0$
9. Window-mean construction of non-dominated solutions and sequential Monte Carlo estimation
10. Calculate R^M and store $(s,a,r,s')^M$ in the experience pool
11. Calculate $L(\theta)^m$ to update the upper-level network
12. Select action a^*, calculate HV, and clear the window
13. end if
14. Calculate $L(\theta)^w$ and update the lower-level network
15. if $g\ mod\ 1000 = 0$ then $\theta^{W-} \leftarrow \theta^W$;$\theta^{M-} \leftarrow \theta^M$
16. if done then break;
17. end for
18. Record the current indicator value as $\mathrm{HV}_s(A)$ and save the optimal model.
19. end for

3 Experimental Design and Result Analysis

This section describes the experimental environment, evaluation metrics, dataset, and baseline algorithms, and then evaluates the proposed method under different resource conditions.

3.1 Experimental environment and setup

The experiments run on an Intel Core i7-12800HX CPU at 2.00 GHz with an NVIDIA GeForce RTX 4070 GPU (8 GB), under Python 3.6 and PyTorch 1.0. A dynamic-load scenario is built from the Alibaba Cluster Trace v2018 [27], which records 40,000 machines over eight days; one day of data is used, and task attributes are extracted in the form of T_i. The correspondence between servers S and virtual machines VM, together with their parameters, is given in Tables 2 and 3.

Table 2 Server types and VM composition

Server category	VM composition
I	A,B,B,C
II	A,A,B,C
III	B,B,C
IV	A,A,D,E

Table 3 Virtual machine configurations

Virtual machine category	CPU	Memory	Static power	Dynamic power factor
A	8.0	0.6	45	7
B	10.0	0.8	63	10
C	10.0	1.2	69	10
D	8.0	1.6	66	9
E	10.0	2.0	74	11

The default settings are as follows: each VM queue holds at most 5 tasks, a task waits at most 40 steps, and training uses the Adam optimizer with ReLU activation. Each step submits 64 tasks sampled from the dataset. Both networks contain four layers: an input layer, two 128-dimensional fully connected layers, and an output layer.

The remaining parameters are listed in Table 4. Performance is evaluated on task completion time, resource utilization, and energy consumption, together with the HV increment and the cumulative reward.

Four baselines are used: round robin (RR) and REINFORCE [28-29] as non-DRL baselines, and DQN and D3QN as DRL baselines. All DRL methods share the hyperparameters in Table 4 and are trained on the same data for the same number of episodes.

Table 4 Algorithm parameter settings

Parameter Name	Parameter value
Discount factor γ	0.99
Gradient clipping threshold	10.0
Upper learning rate	1×10^{-4}
Lower learning rate	5×10^{-5}
Exploring the attenuation rate	0.995
Confidence coefficient z	1.96
Standard error threshold ε	0.02
Iteration round e	1500

3.2 Experimental Results and Analysis

3.2.1 The influence of the learning rate

The learning rate is examined first. Figure 3 shows the cumulative reward obtained with learning rates of 0.001, 0.0001, and 0.00001 [30]. A rate that is too high transmits high-variance noise from the oscillating upper and lower policies to the upper layer and traps the system in a local optimum; a rate that is too low makes weight updates negligible and leads to underfitting. A rate of 0.0001 is therefore adopted, because it allows the lower layer to make fine multi-objective trade-offs while still providing the upper layer with a clear and accurate hypervolume-increment signal.

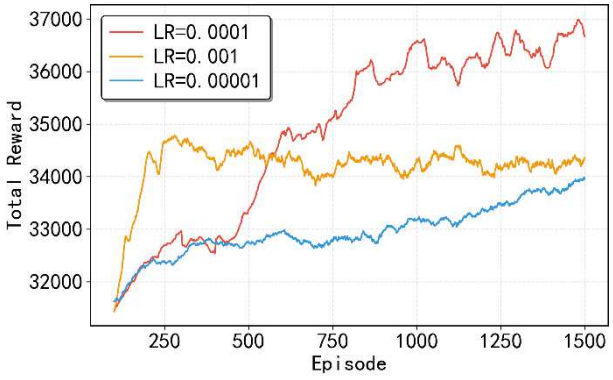


Figure 3 Cumulative reward under three learning rates

Holding the learning rate fixed, Figure 4 compares the reward of the proposed method with that of DQN and D3QN. DQN improves efficiency early in training, but repeated sampling of similar transitions amplifies the oscillation of value estimation, which exposes its instability under complex constraints. D3QN, which combines double Q-learning with a dueling architecture, converges fastest in the early stage, yet fixed weights confine it to a multi-objective bottleneck at approximately 650 episodes.

ITS-D3QN converges slightly later, but its global search capability is clearly stronger: it escapes the local optimum, keeps improving around episode 800, and finally stabilizes at the highest reward.

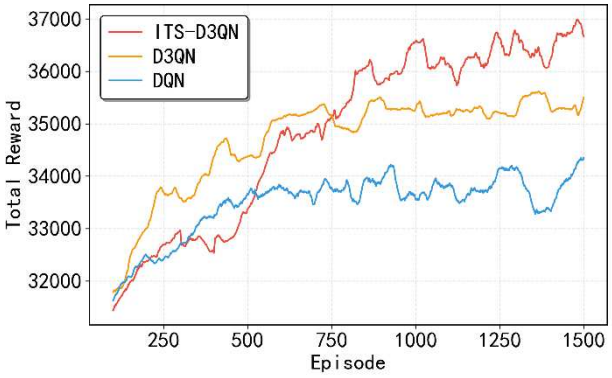


Figure 4 Cumulative reward of the three DRL algorithms

3.2.2 Controlled experiment

The algorithms are then compared at four task volumes: 5,000, 10,000, 15,000, and 20,000. Individual objectives are examined first, followed by the aggregate indicators and the HV trajectory of the proposed method.

1) Task completion time

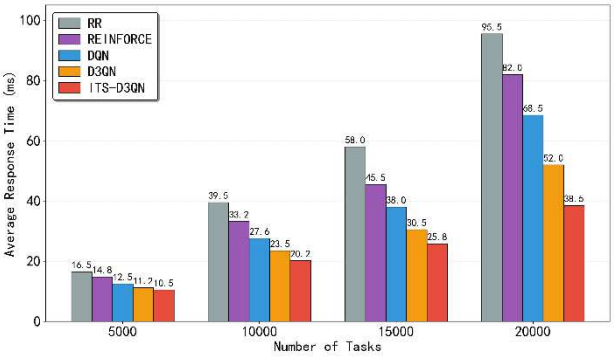


Figure 5 Task completion time of five algorithms under different loads

ITS-D3QN maintains the lowest response time at every scale, reducing it by 29.1% to 59.7% relative to the non-DRL baselines and by 6.3% to 43.8% relative to the DRL baselines, which avoid neither policy oscillation nor fixed weights. The advantage grows with load, indicating robustness under increasing pressure.

2）Resource utilization rate

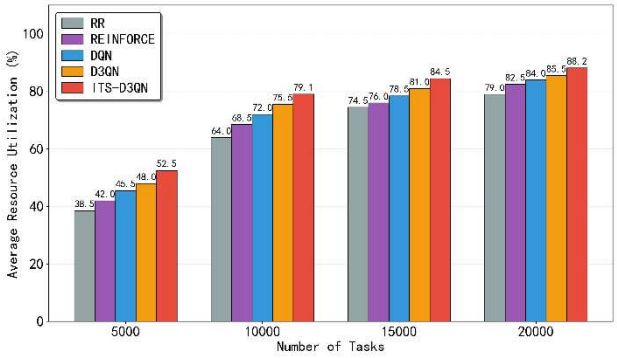


Figure 6 Resource utilization of five algorithms under different loads

With macro-level coordination from the two-layer architecture, ITS-D3QN raises resource utilization by 6.9% to 36.4% over the non-DRL baselines and by 3.2% to 15.4% over the DRL baselines at all task volumes. The gain reflects both reduced idling and better load balancing under tight resources.

3）Energy consumption

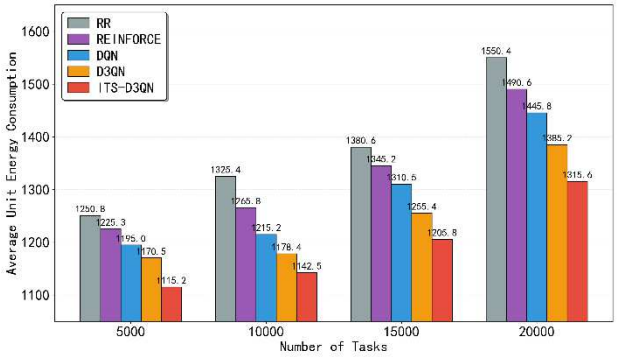


Figure 7 Energy consumption of five algorithms under different loads

Under all load conditions, ITS-D3QN lowers energy consumption by 9.0% to 15.1% relative to the non-DRL baselines, compared with 3.1% to 9.0% for the DRL baselines. This confirms that the method balances high concurrency against energy cost.

4）Comprehensive indicators and HV increment

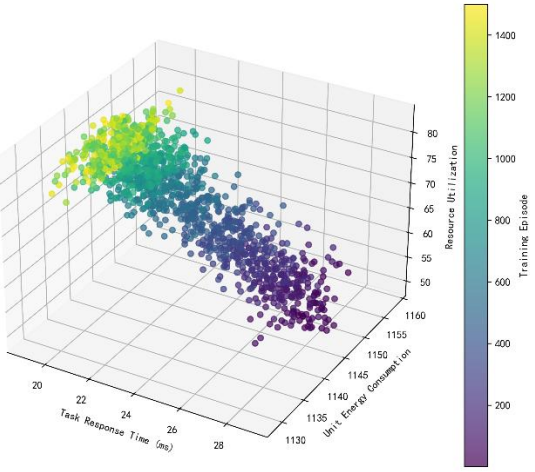


Figure 8 Evolution of the Pareto front during training

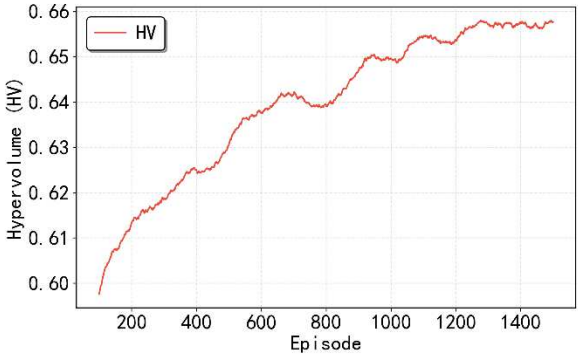


Figure 9 Hypervolume during training

Figure 8 traces the evolution of the proposed method. Early in training, random exploration yields poor schedules. As training proceeds, the solution cloud migrates steadily toward the upper region of the three-dimensional objective space. In the final stage the scheduler reaches a balanced operating point between throughput and energy, sustaining low response time and low per-task energy while raising resource utilization above 80%. The HV curve in Figure 9 confirms that, under conflicting objectives, the method searches effectively toward the multi-objective frontier.

3.2.3 Ablation experiment

Ablation experiments isolate the HV-driven two-layer logic and the incremental sampling. Table 5 shows that removing incremental sampling still improves on D3QN on all three metrics, while the two-layer dynamic weighting produces the larger gain; adding error-controlled sampling removes residual noise and stabilizes the improvement.

Table 5 Ablation results of the ITS-D3QN algorithm

Algorithm variant	Task completion time	Resource utilization rate	Energy consumption
D3QN	23.5	75.5%	1178.4
ITS-D3QN (Incremental sampling is not used)	22.8	78.8%	1151.9
ITS-D3QN	20.2	79.1%	1142.5

4 Conclusion

This paper addresses the Pareto bottleneck and the policy oscillation that arise in multi-objective cloud task scheduling, and proposes a DRL scheduler driven by a Pareto optimization strategy. A two-layer interactive architecture decouples dynamic weight selection from the task placement decision. A macro-reward feedback mechanism based on the hypervolume increment guides the evolution of the Pareto front, overcoming the tendency of conventional scalar reward functions to settle at local optima. An incremental sequential Monte Carlo scheme with confidence-interval testing controls sampling error [31], suppressing noise under highly dynamic load and stabilizing the policy under complex constraints.

Simulations across several task volumes show that ITS-D3QN outperforms the baselines in response time, resource utilization, and per-task energy consumption. It switches scheduling preferences according to system load and balances throughput against energy in high-concurrency scenarios. Future work will extend the Pareto strategy to cloud-native environments with delay jitter and link failures, and will incorporate task priority and service-level agreement guarantees [32].

References

[1] WANG Z, et al. A tail latency SLO guaranteed task scheduling scheme for user-facing services[J]. IEEE Transactions on Parallel and Distributed Systems, 2025.

[2] DENG S G, ZHAO H L, HUANG B B, et al. Cloud-native computing: A survey from the perspective of services[J]. Proceedings of the IEEE, 2024, 112(1): 12-46.

[3] MAO H, ALIZADEH M, MENACHE I, et al. Resource management with deep reinforcement learning[C]//Proceedings of the 15th ACM Workshop on Hot Topics in Networks. 2016: 50-56.

[4] HUANG S, ONTAÑÓN S. A closer look at invalid action masking in policy gradient algorithms[C]//The International FLAIRS Conference. 2022.

[5] FELTEN F, et al. Multi-objective reinforcement learning based on decomposition: A taxonomy and framework[EB/OL]. arXiv preprint, 2024.

[6] RUAN X, et al. Evolutionary multi-objective deep reinforcement learning for computation offloading in IoT[J]. IEEE Internet of Things Journal, 2023.

[7] 陈娟,王阳,吴宗玲,等. 基于深度强化学习的云边协同任务迁移与资源再分配优化研究 [J]. 计算机科学, 2024, 51 (S2): 713-722.

[8] 张玉樟,田乐,魏华利,等. 基于策略熵监督的在线强化学习任务调度算法 [J/OL]. 计算机工程, 1-13[2026-07-22].

[9] 盛煜,朱正伟,朱晨阳,等. 基于深度强化学习的多目标边缘任务调度研究 [J]. 电子测量技术, 2023, 46 (8): 74-81.

[10] NAAZ F, CHINMAY A, PATTANAIK A. QoS aware deep reinforcement learning technique for task scheduling in cloud computing[J]. Discover Computing, 2026, 29(1): 185.

[11] AZARI H Z, JALILI S. Energy-aware self-adaptive real-time job scheduling in a cloud environment using deep reinforcement learning focused on quality of service[J]. The Journal of Supercomputing, 2023, 79(3): 3326-3351.

[12] 叶芳泽,沈炜. 基于深度强化学习改进的任务调度算法 [J]. 计算机时代, 2022, (11): 55-58+64.

[13] 徐东红,李彬,齐勇. 面向云数据中心基于改进A2C 算法的任务调度策略 [J]. 计算机科学, 2025, 52 (2): 310-322.

[14] CHRAIBI A, BEN ALLA S, TOUHAFI A, et al. A novel dynamic multi-objective task scheduling optimization based on Dueling DQN and PER[J]. The Journal of Supercomputing, 2023, 79: 21368–21423.

[15] 王丽萍,任宇,邱启仓,等. 多目标进化算法性能评价指标研究综述 [J]. 计算机学报, 2021, 44 (8): 1590-1619.

[16] 郑金华,李珂,李密青,等. 一种基于 Hypervolume 指标的自适应邻域多目标进化算法 [J]. 计算机研究与发展, 2012, 49 (2): 312-326.

[17] 赵莹莹,仇越,朱天晨,等. 基于分层强化学习的新型电力系统在线稳态调度 [J]. 上海交通大学学报, 2025, 59 (3): 400-412.

[18] LI F, LANG S, TIAN Y, et al. A transformer-based deep reinforcement learning approach for dynamic parallel machine scheduling problem with family setups[J]. Journal of Intelligent Manufacturing, 2025, 36(7): 4735-4768.

[19] XIE Y, GUI F, WANG W, et al. A two-stage multi-population genetic algorithm with heuristics for workflow scheduling in heterogeneous distributed computing environments[J]. IEEE Transactions on Cloud Computing, 2023, 11(2): 1446-1460.

[20] CHRAIBI A, BEN ALLA S, TOUHAFI A, et al. A novel dynamic multi-objective task scheduling optimization based on Dueling DQN and PER[J]. The Journal of Supercomputing, 2023, 79: 21368–21423.

[21] LIU J, ZHANG X. Dynamic computation offloading and resource allocation for mobile edge computing via deep reinforcement learning[J]. IEEE Transactions on Mobile Computing, 2021.

[22] WANG J, ZHAO L, LIU J, et al. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach[J]. IEEE Transactions on Emerging Topics in Computing, 2019, 9(3): 1529-1541.

[23] 孙超利,张澳龙,刘晓彤,等. 一种基于两阶段自适应搜索的云计算任务调度算法 [J]. 计算机应用研究, 2025, 42 (9): 2730-2738.

[24] MENG X, WANG W, ZHANG Z. Delay-constrained hybrid computation offloading with cloud and edge computing[J]. IEEE Access, 2017, 5: 16355-16367.

[25] DAYARATHNA M, WEN Y, FAN R. Data center energy consumption modeling: A survey[J]. IEEE Communications Surveys & Tutorials, 2015, 18(1): 732-794.

[26] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[M]. 2nd ed. Cambridge: MIT press, 2018.

[27] 王小林, 孙莹, 李建江, 等. 面向云数据中心的深度强化学习多目标任务调度算法[J]. 软件学报, 2022, 33(8): 2980-2998.

[28] MOSELEY B, VARDI S. The efficiency-fairness balance of Round Robin scheduling[J]. Operations Research Letters, 2022, 50(1): 20-27.

[29] GREENSMITH E, BARTLETT P L, BAXTER J. Variance reduction techniques for gradient estimates in reinforcement learning[J]. Journal of Machine Learning Research, 2004, 5: 1471-1530.

[30] 黄一帆,曾旺,陈哲毅,等. 移动边缘计算中基于 Actor-Critic 深度强化学习的任务调度方法 [J]. 计算机应用, 2024, 44 (S1): 150-155.

[31] ROSSI A, VISENTIN A, CARRARO D, et al. Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning[J]. Cluster Computing, 2025.

[32] LIN M, XIANG J, CHEN H, et al. Tail latency SLO guaranteed task scheduling scheme for user-facing services in cloud-native environments[J]. IEEE Transactions on Cloud Computing, 2023.