Future Generation Computer Systems 176 (2026) 108174

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

AQESF: An adaptive QoS-enhanced scheduling framework for online batch of task scheduling

Huikang Huang [a], Weiwei Lin [a,b,*], Minxian Xu [c], Keqin Li [d]

[a] Department of Computer Science and Engineering, South China University of Technology, GuangZhou, 510000, China
[b] Pengcheng Laboratory, Shenzhen, 518066, China
[c] Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, 518055, China
[d] Department of Computer Science, State University of New York, New Paltz, NY, 12561, USA

ARTICLE INFO

Keywords:
Cloud computing
Online task scheduling
Deep reinforcement learning
Quality of service

ABSTRACT

For dynamic cloud environments and diverse user requirements, cloud service providers must adopt efficient scheduling methods to fulfill the quality of service (QoS). However, existing scheduling approaches are still inadequate in dealing with the online batch task scheduling problem in complex cloud environments. Specifically, existing methods do not consider the scheduling order optimization of batch tasks while taking into account long-term cumulative performance and robustness. This paper proposes an Adaptive QoS-Enhanced Scheduling Framework (AQESF) based on the multi-action Proximal Policy Optimization to address this challenge. The AQESF integrates the Deep Reinforcement Learning (DRL) Queue and the Multi-FIFO-Manner modules for joint optimization to cover the task order and task placement solution space. Furthermore, placement decisions are constrained to be solved in a more optimized space based on well-designed greedy algorithms. Extensive experimental evaluations on the Alibaba trace demonstrate that AQESF exhibits superior cumulative performance of average response time and success rate. Furthermore, AQESF exhibits strong robustness and low scheduling latency compared with the common DRL task scheduling paradigm. Finally, we analyze the potential applications of AQESF in VM placement and computation offloading.

1. Introduction

1.1. Background

With the high reliance of various enterprises on the convenient and fast cloud computing service paradigm, cloud computing has been treated similarly to public utilities (water, electricity, etc.). Users can subscribe to cloud services tailored to their specific requirements [1]. Cloud service providers (CSPs), drawing upon the hardware resources in cloud data centers, employ virtualization technology to consolidate and integrate heterogeneous hardware devices into virtual resource pools [2] to provide users with safe and reliable remote computing and storage services. Furthermore, to deliver QoS to users and enhance resource utilization, CSPs need to manage data center resources by leveraging scheduling techniques.

For CSPs, deploying a stable and reliable scheduling method is a prerequisite for realizing operational benefits. Furthermore, proposing advanced scheduling methods to improve revenue is compelling but challenging. Overall, the design of scheduling methods needs to consider two essential points:

- *Stability*. In dynamic cloud environments, not only do workload requests change, but accidents are inevitable within the data center, such as server downtime. The scheduling method needs to be robust enough to handle these unforeseen events.
- *Superiority*. The task scheduling problem behaves as an optimization problem with non-deterministic polynomial time complexity, i.e., NP-hard. Therefore, sub-optimal scheduling decisions should be as close as possible to the optimal solutions. Simultaneously, it is crucial to take into account long-term benefits.

In general, the data center receives batch tasks from users and then schedules all tasks simultaneously by Broker [3,4], such type of problem is abstracted as the generalized Batch-of-Tasks (BoT) scheduling problem in this paper. Further, we consider the BoT online scheduling problem (*No main cache queue, all tasks need to be scheduled at current time*).

* Corresponding author.
E-mail addresses: huikanghuang0321@gmail.com (H. Huang), linww@scut.edu.cn (W. Lin), mx.xu@siat.ac.cn (M. Xu), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.future.2025.108174
Received 6 August 2025; Received in revised form 21 September 2025; Accepted 27 September 2025
Available online 1 October 2025

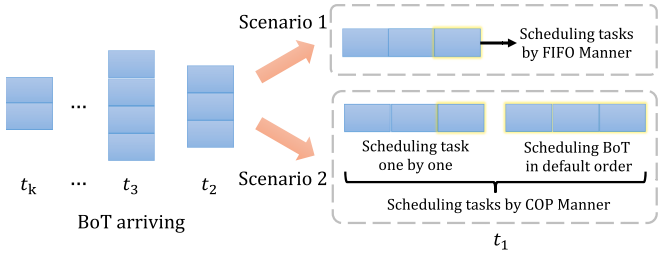


Fig. 1. Two modeling paradigms and solving manners.

There are generally two scenarios modeling and solution methods, as shown in Fig. 1:

- **Scenario 1.** The data center will take the tasks that arrive simultaneously, randomly put them into a FIFO queue, and then schedule them sequentially according to the order of the tasks in the queue. This is to model the BoT as a FIFO task queue, only considering the information of the first task in the queue that currently needs to be scheduled, which is equivalent to scheduling one task online. Algorithms that handle this scenario are MCT, E-PVM [5], and also Deep Reinforcement Learning (DRL) [6–8].
- **Scenario 2.** Modeling simultaneous arriving tasks as a Combinatorial Optimization Problem (COP), and tasks within the BoT are scheduled one by one, e.g., Min-Min. More directly, the entire BoT is mapped for scheduling in a default order, e.g., Genetic Algorithms (GA) and Ant Colony Systems [9–11] or DRL to solve COP [12]. This modeling approach considers the information of the whole BoT, different from Scenario 1 (only one task).

1.2. Challenges and contributions

For the online BoT scheduling problem, the main challenge that exists is how to consider scheduling decisions to optimize the long-term benefits with good robustness. However, the above two modeling paradigms and solving methods are difficult to meet the requirements of long-term benefit and decision-making stability simultaneously.

Specifically, for optimizing the solution space coverage problem, the modeling paradigm of **Scenario 1** inevitably fails to cover the current BoT's own solution space, ignoring potential optimizations brought about by task scheduling order. For **Scenario 2**, both types of algorithms can obtain BoT information for decision-making. Algorithm type 1 can optimize the scheduling order of tasks, while type 2 adopts the default order. Nevertheless, neither of the two algorithm types can take into account the impact between the decision at t and $t+1$, i.e., optimization for sequential decision-making problems.

Regarding algorithm design, heuristics are highly explainable and controllable, and their suboptimal scheduling results are within acceptable margins. However, heuristic algorithms focus on optimizing task scheduling at the current time or current batch, focusing on immediate gains and neglecting to optimize continuous incoming task requests in the long run. DRL is a promising approach for solving the long-term benefits. However, the stability of the environment is a key prerequisite for the effective training and decision-making capability of DRL. In the case of drifting data distribution, DRL-based scheduling models may fail due to insufficient generalization. In addition, when using DRL to solve the online BoT scheduling problem, the optimal solution cannot be obtained if the problem modeling does not cover the optimal solution space. Unfortunately, current research on online BoT scheduling optimization based on DRL cannot effectively meet these requirements. Specifically, it fails to simultaneously consider algorithmic robustness and coverage of the solution space (scheduling order and placement) while optimizing long-term rewards.

In this paper, we focus on the basic scheduling entity BoT, i.e., scheduling multiple BoTs to optimize the average response time (ART) and success rate (SR). This problem is formulated as a continuous decision problem embedded with COPs. We propose a novel Adaptive QoS-Enhanced Scheduling Framework (AQESF) based on multi-action Proximal Policy Optimization (PPO) to optimize the cumulative performance. Briefly, several significant contributions of this paper can be summarized as:

1. We formally define the online BoT scheduling problem based on ART and SR optimization under resource constraints and prove that the problem is NP-hard in a static situation. Considering cumulative reward optimization in the dynamic cloud environment, we model the problem as a Markov decision process (MDP) and define the corresponding state, action and reward function.
2. We propose AQESF, which integrates DRL-Queue and Multi-FIFO-Manner (Multi-FM) modules to cover the solution search space for task ordering and placement of the online BoT scheduling problem, further approximating the optimal solution while considering long-term rewards.
3. We design heuristic algorithms Minimum Completion Time (MCT) and Opportunity Load Balancing (OLB) within Multi-FM, and adaptively select one of them according to DRL to implement scheduling decisions in white-box scenarios, thereby limiting the decision-making to a controllable and more optimal space.
4. By conducting extensive experiments on the Alibaba trace, we compare with well-designed algorithms in different types (traditional heuristics, meta-heuristics, DRL-based algorithms). Our findings demonstrate that AQESF can improve the heuristic algorithm to achieve QoS-enhanced. Moreover, AQESF provides more robustness and lower scheduling delay than the widely used DRL scheduling paradigm.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 gives the problem definition and analyzes the complexity of the problem. We focus on our AQESF in Section 4. Section 5 validates our method through extensive experiments. Section 6 gives potential applications of our method in two application scenarios. We summarize the work of this paper in the final section.

2. Related work

Generalized BoT scheduling research covers a variety of application scenarios. It can be reduced to bin packing, job shop scheduling or sequential decision problems in most cases. Existing studies mainly use approximation algorithms for problem-solving, such as traditional heuristics (greedy strategies, rule-based methods), meta-heuristics (iterative search to obtain the optimal decision), and DRL methods.

2.1. Heuristics methods

In recent years, there have been emerging relevant research findings for traditional heuristic methods. It is widely known that well-designed heuristics provide acceptable scheduling performance, stability, interpretability, and low scheduling delay. For example, Gupta et al. [13] designed an enhanced Min-Min algorithm to optimize the completion time of job flow and achieve load balancing. Wang et al. [14] designed Sensible, an adaptive algorithm based on a rule-based policy that constantly updates the priority of each host through ongoing measurements of task RT. To address the Bi-objective optimization problem of scheduling parallel jobs online to maximize user satisfaction and SaaS revenue, Zheng et al. [15] proposed a greedy policy with $\delta/\theta(\rho-2)$ competitive ratio to decide the purchase of instances and scheduling jobs. Constructing VM scheduling as an online task scheduling problem, Lin et al. [16] proposed PEAS, a scheduling method based on optimal server energy efficiency, which aims to select the PM with the most resource-satisfying and energy-efficient PM for VM placement.

For COPs composed of a batch of workloads and system states, using meta-heuristic algorithms enables the search for solutions based on

the designed fitness function. Although meta-heuristic methods perform better in multi-objective optimization, they are generally only suitable for latency-tolerant workloads due to the long solving time. Kumar et al. [17] proposed a PSO-BOOST scheduling algorithm that gives a non-dominated optimal solution set between cost and execution time based on Pareto optimality theory. Sun et al. [18] proposed a flexible and lightweight genetic algorithm (FGA) based on a polysomy-strengthening elitist genetic algorithm to minimize the weighted sum of time and energy consumption in dynamic task scheduling problems. Zhang et al. [9] proposed an improved multi-objective optimization method, In-MaOEA, which can simultaneously take into account task completion time, scheduling cost, and task completion rate. Mousavi et al. [19] proposed a directed non-dominated sorting genetic algorithm called D-NSGA-II by introducing a recombination operator in NSGA-II. The algorithm can control the selection pressure of the intelligence and use the new operator to balance the exploratory and exploitative capabilities of the algorithm. Considering the optimization of energy consumption and makespan of MapReduce tasks, Demirbaga et al. [20] construct the scheduling problem as a COP and propose the Ant Colony Optimisation to solve it.

2.2. DRL-based methods

In recent years, DRL has been proposed for task scheduling to learn the optimal decision through agent-environment interaction to optimize cumulative rewards. Tong et al. [6] proposed a DDQN-TS task scheduling scheme with a Bi-objective weighted reward function designed to optimize the task completion rate and ART. Considering that different types of jobs have different processing speeds on different types of virtual machines (VMs) (I/O-intensive and computation-intensive), Huang et al. [21] relied on predefined expert knowledge to optimize the ART and the SR of latency-sensitive jobs using Dueling-DQN based on a generative adversarial imitation learning framework. Similarly, Tang et al. [22] considered the task scheduling problem for heterogeneous VMs in multi-cloud environments and proposed cost and makespan aware DQN scheduling algorithms to train task scheduling by weighting cost and makespan. Cui et al. [23] designed a multi-agent parallel Q-learning algorithm to minimize the makespan and average waiting time of jobs. Cheng et al. [24] considered VM rental cost and QoS (ART and SR) in hybrid cloud environments. They used DQN to optimize the two objectives for weighted rewards. The experimental results show that the proposed method has a significant advantage in terms of cost, but still does not perform as well as some traditional heuristics in terms of ART. Staffolani et al. [7] considered the workload allocation for distributed queues. They proposed an RLQ scheduling framework based on the Double DQN and the contextual bandit. They demonstrated that the RLQ has fast convergence properties and better performance through extensive experiments. Yang et al. [4] modeled the cost and benefit of batched tasks as a single-objective optimization problem and proposed a DRL + greedy scheduling method to maximize the data center gain. However, the greedy algorithm makes it difficult to deal with problems that do not have an optimal substructure in the time-series dimension.

2.3. Summary

The two major classes of algorithms (Heuristics and DRL-based methods) are commonly used to deal with task scheduling problems, but all the existing studies have limitations. For example, heuristic methods cannot consider the optimization of cumulative performance, which fails to obtain cumulatively optimal solutions for problems that do not have optimal substructures in the time-series dimension. For online BoT scheduling problems, if modeled as the scheduling paradigm in **Scenario 1**, using DRL to solve the problem results in insufficient state-action space coverage (i.e., without considering the task order). If modeled as **Scenario 2**, the optimization of long-term rewards cannot be considered. Additionally, relying on DRL for global optimization search can lead to insufficient robustness. Therefore, the aforementioned scheduling methods still have some shortcomings.

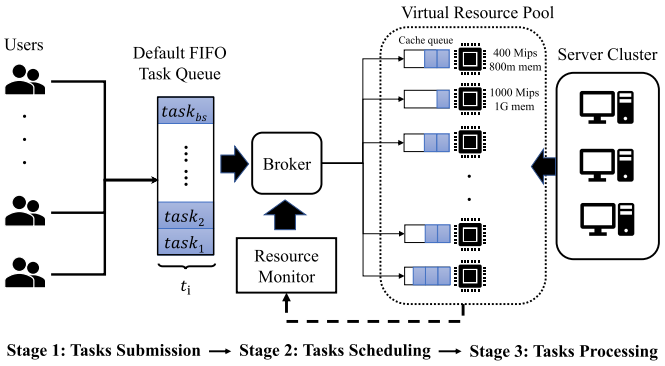


Fig. 2. Prototype of BoT scheduling in cloud data center.

3. Problem formulation

3.1. System model

To better understand the problem studied in this paper, we show the problem prototype in Fig. 2. Specifically, the generalized BoT scheduling procedure can be divided into the following stages:

1. **Tasks Submission**. Multiple users initiate tasks with resource requirements (e.g., task length, task type, memory usage, etc.) to the cloud data center. In this study, we do not consider the main cache queue, i.e., all tasks received at the current time need to be scheduled to the VM cache queue through the Broker and cannot be postponed to the next time, i.e., online scheduling.
2. **Tasks Scheduling**. Based on the information collected by the resource monitor, the Broker orchestrates the online scheduling of tasks to VMs within the designated resource pool using pre-established scheduling algorithms. Typically, two categories of algorithms are employed, as shown in Fig. 1. The FIFO-manner algorithm schedules tasks sequentially without using information from other tasks in the current batch (e.g., MCT, round-robin, etc.). And the COP-manner algorithm schedules a BoT concurrently (e.g., Min-Min, GA, etc.).
3. **Tasks Processing**. When sufficient resources are available, tasks can be executed immediately. Otherwise, tasks will wait for execution on the VM cache queue, similar to many existing works [6,7]. In this paper, we do not consider task backfilling and rescheduling. We only consider FIFO queues in VM cache queue to ensure absolute fairness.

3.2. Problem definition

The problem studied in this paper is formally defined in this subsection. First, we describe the frequently used symbols in Table 1. Then, a detailed modeling description is provided for this problem.

The CSP can receive user task requests and monitor resource usage in the data center, which is the problem's input. For each task request, represented by a tuple: $task_j =< task^{id}, task^{arrT}, task^{len}, task^{mem}, task^{ect} >$. Where $task^{id}$ is the unique number of the task, $task^{arrT}$ is the arrival time, $task^{len}$ is the length of the million instructions that need to be processed, $task^{mem}$ is the memory requirement, and $task^{ect}$ is the expected completion time. For data center resources, VMs serve as compute nodes for task execution, represented by: $VM_i =< VM^{id}, VM^{proc}, VM^{mem}, VM^{idleT} >$. Where VM^{id} is the identification of VM, VM^{proc} denotes the processing speed of VM, and VM^{mem} is the memory capacity. VM^{idleT} denotes the time required for the VM resource release to transition to the idle state.

Table 1
List of symbols.

Name	Description
n	The number of tasks
m	The number of VMs
bs	The batch size of tasks
$task^{id}$	The id of task
$task^{arrT}$	The arrival timestamp of task
$task^{len}$	The million instruction length of task
$task^{mem}$	The memory of task request
$task^{ect}$	The expected completion time of task
$task^{startT}$	The start timestamp of task
$task^{endT}$	The finish timestamp of task
$task^{sr}$	The successful symbol of task
VM^{id}	The id of VM
VM^{proc}	The processing speed of VM
VM^{mem}	The memory of VM
VM^{idleT}	Time required for VM to release resources into idle
s_t	The system state at timestamp t
a_t	The action selected by DRL agent at timestamp t
r_t	Total rewards at timestamp t
RT	The response time of task
SR	The success rate of total tasks

Similar to the existing works [21,23], we assume that each task request is independent and cannot be further partitioned for processing. Each task can be executed by only one VM, and each VM can only process one task request at a time, which is the same as the Space-Shared mode in the well-known simulation framework Cloudsim [25]:

$$x_{i,j} = \begin{cases} 1, & task_j \ is \ placed \ in \ VM_i \\ 0, & otherwise, \end{cases} \tag{1}$$

where $x_{i,j}$ means that $task_j$ is placed on VM_i or waiting for execution in VM_i's cache queue. Tasks will wait for execution in the FIFO queue. Then, after $task_j$ is scheduled to VM_i, the start time $task_j^{startT}$ of $task_j$ is expressed as:

$$task_j^{startT} = max(0, VM_i^{idleT}) + task_j^{arrT}. \tag{2}$$

The QoS metrics under consideration in the task scheduling problem include ART and SR. ART signifies the mean value of all task response times within the observation window, directly reflecting user satisfaction. It is expressed as follows:

$$ART = \frac{1}{n}\sum_{j}^{n} RT(task_j). \tag{3}$$

Moreover, SR is intricately linked to the Service Level Agreement (SLA), signifying whether the RT of a task can be accomplished within a predefined $task^{ect}$ during the observation window. $task^{ect}$ is specified in the User-CSP agreement, and the violation will penalize the data center. Therefore, we designate SR as the optimization objective at the data center side, which is expressed by:

$$SR = \frac{1}{n}\sum_{j}^{n} task_j^{sr}. \tag{4}$$

We address the task scheduling problem under resource constraints, which can be formulated as an Integer Programming problem. It is imperative to highlight that, within the time-series dimension, scheduling decisions have a subsequent impact on the system state once tasks are heavy at the current time, i.e., the current BoT could not be completed when the next BoT arrives. Therefore, the problem is also a sequential decision problem, which can be expressed as follows:

$$min(ART) = min\frac{1}{n}\sum_{B=bs_1,\dots,bs_t}\sum_{i}^{m}\sum_{j}^{B} x_{i,j} RT(task_j) \tag{5}$$

$$max(SR) = max\frac{1}{n}\sum_{B=bs_1,\dots,bs_t}\sum_{i}^{m}\sum_{j}^{B} x_{i,j} task_j^{sr} \tag{6}$$

s.t.

$$RT(task_j) = task_j^{startT} - task_j^{arrT} + \frac{task_j^{len}}{VM_i^{proc}}, \tag{7}$$

$$task_j^{sr} = \begin{cases} 1, & RT(task_j) \le task_k^{ect} \\ 0, & otherwise, \end{cases} \tag{8}$$

$$x_{i,j} = \begin{cases} 1 \ or \ 0, & task_j^{mem} \le VM_i^{mem} \\ 0, & otherwise, \end{cases} \tag{9}$$

$$bs_1 < bs_2 <, \dots, < bs_t, \tag{10}$$

$$x\sum_{i}^{m} x_{i,j} = 1, \forall j \in n, \tag{11}$$

$$x_{i,j} \in \{0,1\}, \forall i \in m, j \in n, \tag{12}$$

Eqs. (5) and (6) are the optimization objectives of the problem. Eq. (7) gives the calculation of the task RT. Constraint Eq. (8) implies that the RT of a task needs to be completed within a pre-defined $task^{ect}$ to be valuable. Constraint Eq. (9) means that tasks can only be assigned to VMs with sufficient resources. Constraint Eq. (10) indicates that different batches of tasks are to be scheduled in temporal order. Eqs. (11) and (12) represent that all tasks need to be scheduled and executed.

3.3. Problem analysis

Intuitively, the optimal scheduling decision for the current BoT can be obtained by enumeration, but it takes exponential time. The following illustrates that the BoT scheduling problem, explicitly focusing on SR optimization, is NP-hard.

Theorem 1. *In resource-constrained scenarios, online scheduling for the current BoT to optimize SR is an NP-hard problem.*

Proof. To prove Theorem 1, we reduce this task scheduling problem to the Multiple Knapsack Problem (MKP), which has been proven to be NP-hard [26]. In this reduction, tasks and VMs are considered items and multiple knapsacks. Scheduling is performed for the BoT at the current time, so Eqs. (6) and (10) take only one time slot b_i, i.e., only one BoT is considered for SR. Alternatively, in the simpler case, we set knapsacks to be empty, i.e., Eqs. (7) and (8) can be further simplified and merged. Then, the problem can be reduced to:

$$max(SR) = max\frac{1}{n}\sum_{i}^{m}\sum_{j}^{bs} x_{i,j} task_j^{sr} \tag{13}$$

s.t.

$$\sum_{j}^{bs} x_{i,j} task_j^{len} \le VM_i^{proc}, \forall i \in m, \tag{14}$$

$$\sum_{i}^{m} x_{i,j} = 1, \forall j \in n, \tag{15}$$

$$x_{i,j} \in \{0,1\}, \forall i \in m, j \in n, \tag{16}$$

The static scheduling of the current BoT can be viewed as a complex unfolding of the MKP. Consequently, the problem studied in this paper is also NP-hard. □

However, we study the problem of online scheduling for BoT in dynamic scenarios. The objectives ART and SR are both cumulative performances, i.e., evaluation metrics computed after the completion of scheduling of all tasks within the observation window. The problem can be represented as a typical MDP in the time-series dimension. This implies the sequential decision problem incorporating the COP, where the COP is proved to be NP-hard in Theorem 1. Therefore, the modeling paradigm of the problem is described in Fig. 3, the action a_t needs to maximize the immediate reward of the current COP as well as optimize the cumulative reward on the time-series dimension.

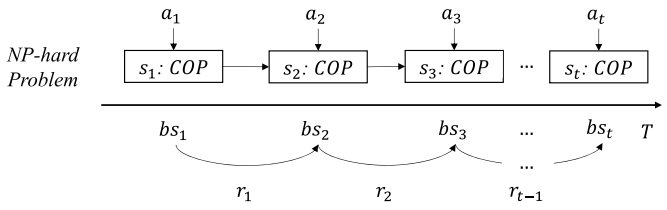


Fig. 3. Sequential Decision Problem incorporating Combinatorial Optimization Problem.

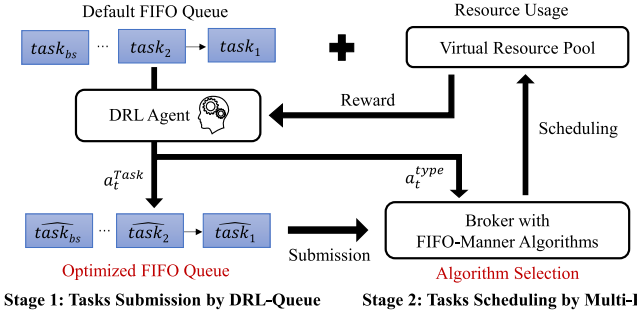


Fig. 4. Adaptive QoS-Enhanced Scheduling Framework.

4. Task scheduling by adaptive QoS-enhanced scheduling framework

4.1. Adaptive QoS-enhanced scheduling framework

For the problem prototype illustrated in Fig. 3, an intuitive approach is to optimize the cumulative rewards using DRL techniques, i.e., scheduling each BoT to optimize immediate reward at each time. However, outputting a BoT scheduling decisions based on DRL requires consideration of task scheduling order (with $bs!$ search space) as well as task placement locations (with m^{bs} search space). Although DRL can learn both of them simultaneously, the search space for solutions is very large (i.e., $m^{bs} \times bs!$), resulting in slow convergence or even failure to converge. In addition, once the environment changes or the data distribution drifts, the model becomes more fragile and uncontrollable because task placement decisions are made in a black-box environment, leading to a serious decline in scheduling performance.

Therefore, to compress the search space and provide a certain scheduling stability, we propose a new method for online BoT scheduling and give an Adaptive QoS-Enhanced Scheduling Framework as shown in Fig. 4. AQESF has two co-optimization modules, named DRL-Queue and Multi-FIFO-Manner. AQESF implements the joint optimization of these two modules based on a multi-action DRL, optimizing both the cumulative reward as well as the scheduling robustness.

DRL-Queue. When the system receives multiple tasks $\{task_1, task_2, \ldots, task_{bs}\}$ simultaneously (i.e., BoT), the system generally gets a random FIFO queue for task scheduling: $task_1 \leftarrow task_2 \leftarrow \ldots \leftarrow task_{bs}$. DRL-Queue is a scheduling order decision maker based on multi-action DRL for outputting an optimized FIFO queue for task scheduling: $\hat{task}_1 \leftarrow \hat{task}_2 \leftarrow \ldots \leftarrow \hat{task}_{bs}$. Therefore, based on the DRL-queue, we can obtain any task scheduling order in a BoT.

Multi-FM. Then, based on the FIFO-manner scheduling algorithm, the tasks are assigned to VMs one by one for processing or waiting in the VM cache queue. Considering that only one greedy algorithm cannot be adapted to dynamic cloud environments (illustrated by the Example 1 shown in Fig. 5), we design a DRL-driven scheduling algorithm selection module Multi-FM. It adaptively selects FIFO-manner scheduling algorithms from $type$ preset algorithms.

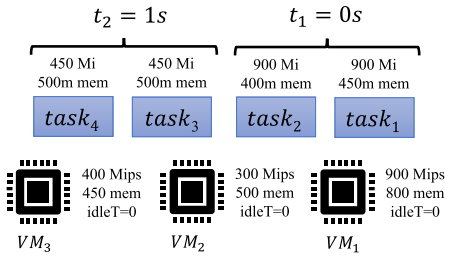


Fig. 5. The greedy algorithm cannot obtain a cumulative optimal solution for the problem without optimal substructures.

Obviously, DRL can be used to train two modules simultaneously to achieve joint optimization of long-term rewards, reducing the search space from $(m^{bs} \times bs!)$ to $(type \times bs!)$. Although it may not cover the optimal search space, scheduling decisions can be restricted to a suboptimal search space, with FIFO-manner algorithms providing performance guarantees. Meanwhile, this means that scheduling decisions are made in a white-box environment, where scheduling decisions can be captured and interpreted. Ultimately, AQESF is able to achieve both *superiority* and *stability*.

Example 1. Suppose we can schedule tasks arriving at the same time in any order based on DRL-Queue, but only one type of greedy algorithm can be used, as illustrated in Fig. 5. When the MCT algorithm is selected for scheduling, we are able to get $ART = (1+2)/2 = 1.5$ at t_1 by $(task_1 \rightarrow VM_1, task_2 \rightarrow VM_1)$, which is the optimal decision at t_1. At t_2, we can get $ART = (1+2+1.5+1.5)/4 = 1.5$ still by MCT $(task_3 \rightarrow VM_2, task_4 \rightarrow VM_1)$. However, using the ORACLE scheduling policy, after scheduling at t_1 $(task_1 \rightarrow VM_1, task_2 \rightarrow VM_3)$ and t_2 $(task_3 \rightarrow VM_1, task_4 \rightarrow VM_1)$, we can get $ART = 1.1875$, reducing the cumulative ART by about 26% compared to the MCT algorithm alone. This example shows that a single greedy algorithm cannot obtain optimal cumulative rewards on scheduling problems that do not have optimal substructures in the time-series dimension because of resource constraints and dynamic requests. Furthermore, it illustrates that the deterministic search strategy of a single greedy algorithm limits the scope of solutions.

4.2. Scheduling problem as MDP and algorithm design

4.2.1. MDP for decision-making

This paper addresses the online scheduling problem for BoT, which is inherently a sequential decision problem. MDP is a general method for modeling, which can be represented by a tuple: $M =< S, A, P, R >$. Where $S = \{s_t\}_1^T$ denotes a set of states following the system, and s_t represents the state at the current time. At each time t, the system chooses an action a_t, $a_t \in A$, and applies it to the environment. The system state will update according to the state transfer function $p(s_{t+1}|s_t, a_t)$, $p \in P$, then receives a reward r_t, $r_t \in R$ and moves to the next time state s_{t+1}. After traversing all tasks, the complete trajectories are obtained as follows:

$$s_1, a_1 \rightarrow r_1, s_2, a_2 \rightarrow \ldots \rightarrow r_t, s_{t+1}. \tag{17}$$

However, the MDP state transfer probability P is unknown, and the whole system has a continuous state and action space. Therefore, we use the model-free PPO [27] for the problem solution. The corresponding state space, action space, state transfer probability, and reward function are defined below.

4.2.2. Description of DRL

State space. Regarding the design of the state space S, there are two components: the messages carried by tasks S^{Task} and the resource usage of the data center S^{VM}. Specifically, at the current time t, the BoT with the task state s_t^{Task} includes $Task^{len}$ and $Task^{mem}$. The resource state

s_t^{VM} is the time VM^{idleT} needed for VM resources to be released. Then the state s_t at time t can be expressed as:

$$\begin{aligned} s_t &= [s_t^{VM}, s_t^{Task}] \\ &= [VM_1^{idleT}, \ldots, VM_m^{idleT}, task_1^{res}, \ldots, task_{bs}^{res}], \end{aligned} \tag{18}$$

where $res \in \{len, mem\}$, denoting the two resources of tasks.

Action space. For the action space A, we use a continuous variable as the task score for the DRL-Queue module, ranging from $[0, 1]$. The score will determine the order of this BoT in the FIFO queue simultaneously. In addition, we integrate Multi-FM for joint optimization, using continuous type variables a_t^{type} to determine the choice of the FIFO-manner scheduling algorithm. The action a_t is shown below:

$$a_t = [a_t^{Task}, a_t^{type}] = [a_t^{task_1}, a_t^{task_2}, \ldots, a_t^{task_{bs}}, a_t^{type}], \tag{19}$$

where $a_t^{task_i}$ is the score of the current BoT. a_t^{type} is the value to represent the selection of scheduling algorithms, divided equally between $[0, 1]$ according to the number of candidate scheduling algorithms.

For the variable-length bs problem, we use large enough s_t and a_t to cover the state and action space of the largest bs of BoTs. For small bs, the state will fill with 0, and the masking action [28] will be adopted to deal with the redundant a_t, which is widely used in natural language processing to handle different lengths of sentences.

State-transition probability. Since the agent cannot obtain the state transition probability $P(s_{t+1}|s_t, a_t)$ of the whole system, we based on the model-free PPO to learn the transfer probability $P : S \times A \to S$, as well as the maximum cumulative reward action a_t through the continuous interaction between the DRL agent and environment.

Immediate reward. For the design of the reward function, some correlation between ART and SR can be found. Specifically, when optimizing SR for the current batch of BoT, it ensures that ART will not be too bad. However, optimizing ART does not necessarily lead to significant improvements in SR. Therefore, the reward is defined as SR within AQESF, which optimizes the NP-hard problem proven by Theorem 1:

$$r_t = \frac{1}{bs} \sum_{bs} task_j^{sr}. \tag{20}$$

It is also possible to use a reward function weighted by ART and SR [21]. But adding parameters reduces the usability of the algorithm. It is worth noting that DRL is only used to improve the performance of the scheduling algorithm, and the task placement decisions are still determined by the scheduling algorithm. Therefore, we discuss the adopted scheduling algorithm for optimizing ART in the next subsection.

4.2.3. Scheduling algorithm

Algorithms 1 and 2 are the pseudo-codes of AQESF and Multi-FM with candidate scheduling algorithms, respectively. For the optimization objectives (Eqs. (5) and (6)), we design Multi-FM embedding two FIFO-manner algorithms, MCT and OLB, as described in Algorithm 2. The heuristic idea is to improve the utilization of high-performance computing resources, which is intuitively seen in two cases:

1. When there are fewer tasks in recent BoT, the MCT algorithm is used to greedily select the VM with the shortest RT for scheduling, which can improve the ART of tasks.
2. When there are more tasks in the recent BoT, and the demand for heterogeneous resources is high, we design the OLB algorithm to achieve load balancing and improve the utilization of high-performance VMs.

Both scheduling algorithms can improve ART. MCT is designed to improve the use of high-performance computing resources, while OLB enhances the use of high-performance computing resources on the premise of improving VM utilization. Therefore, the reward is designed to optimize only SR (Eq. (20)). Nonetheless, manually selecting the FIFO-manner scheduling algorithm based on system states proves challenging, involving numerous factors such as current system state, future task requests, and task ordering. Therefore, we designed Multi-FM, which uses DRL to autonomously determine the algorithm (Algorithm 1, lines 7–8), and combines it with the task ordering output by DRL-Queue (Algorithm 1, line 6) to ultimately achieve collaborative optimization.

Time complexity. The time complexity of scheduling BoT consists of two parts. For the DRL-Queue, we set L as the number of layers of the network and L_i as the number of neurons in the ith layer. The

Algorithm 1: AQESF training algorithm.

1 **Input:** Initialize Actor π_{θ_A} with parameter θ_A, $\theta_A^{old} = \theta_A$. Initialize Critic π_{θ_C} with parameter θ_C, $\theta_C^{old} = \theta_C$. Initialize learning rate α_A and α_C, n_{steps}, discount factor γ, clip coefficient ϵ, replay buffer β.
2 **Output:** Optimized parameters θ_A and θ_C.
3 **for** *epoch=1,2,…,N* **do**
4 **for** *t=1,2,…,T* **do**
5 Obtain action a_t by π_{θ_A} to get the scores of tasks based on the state $s_t = [s_t^{VM}, s_t^{Task}]$;
6 Update queue $\{task_1 \leftarrow task_2 \leftarrow \ldots \leftarrow task_{bs}\}$ to $\{ta\hat{s}k_1 \leftarrow ta\hat{s}k_2 \leftarrow \ldots \leftarrow ta\hat{s}k_{bs}\}$ according to a_t^{Task};
7 Choose Scheduling algorithm according to a_t^{type};
8 Apply Algorithm 2 to get r_t;
9 Obtain s_{t+1} and $Store(s_t, a_t, r_t, s_{t+1})$ in β;
10 **if** $len(\beta) > n_{step}$ **then**
11 **for** *update epoch = 1 to K* **do**
12 Sample mini-batch trajectories from β and Compute loss function(22)(25);
13 Update θ_A and θ_C via (24) and (27) with Adam Optimizer, respectively;
14 **end**
15 Clear β;
16 Soft update $\theta_A^{old} = \theta_A$ and $\theta_C^{old} = \theta_C$;
17 **end**
18 **end**
19 **end**

Algorithm 2: Multi-FIFO-manner module with MCT and OLB algorithms.

1 **Input:** a_t^{type},$\{ta\hat{s}k_1 \leftarrow ta\hat{s}k_2 \leftarrow \ldots \leftarrow ta\hat{s}k_{bs}\}$, s_t^{VM}, s_t^{Task}.
2 **Output:** r_t and Update s_{t+1}^{VM}.
3 Reset $r_t = 0$;
4 **if** $a_t^{type} \geq 0.5$ **then** // OLB
5 **for** $j = 1$ *to* bs **do**
6 Select $\{VM\}_{min\_idleT}$ with the minimum idle time which meet $task_j^{mem}$;
7 Select the VM_{fast}^{proc} with the fastest process speed from $\{VM\}_{min\_idleT}$;
8 Scheduling $task_j$ to VM_{fast}^{proc} and obtain $task_j^{sr}$;
9 $r_t += task_j^{sr}$;
10 **end**
11 **end**
12 **else** // MCT
13 **for** $j = 1$ *to* bs **do**
14 Select $\{VM\}$ subset which meet $task_j^{mem}$;
15 Select the VM_{mct}^{proc} with the minimum completion time from $\{VM\}$;
16 Scheduling $task_j$to VM_{mct}^{proc} and obtain $task_j^{sr}$;
17 $r_t += task_j^{sr}$;
18 **end**
19 **end**

time complexity of DRL-Queue is $O(\kappa)$, $\kappa = \sum_{i}^{L-1}(L_i * L_{i+1})$. For the Mutil-FM, the time complexity of the adopted scheduling algorithm is $O(bs)$, and bs denotes the number of tasks in a batch. Hence, for each BoT, the total time complexity is $O(\kappa + bs)$.

4.2.4. Model-training

For the AQESF training, we used the PPO-Clip model. Different from the general DRL model, the PPO-Clip model proposes the objective of a truncated probability ratio to estimate the performance of the Actor in the Actor-Critic network. Where the truncation probability ratio is used for the importance sampling weights of the new Actor network π_{θ_A} and the old Actor network $\pi_{\theta_A^{old}}$:

$$\rho_t(\theta_A) = \frac{\pi_{\theta_A}(a_t|s_t)}{\pi_{\theta_A^{old}}(a_t|s_t)}. \tag{21}$$

For a more stable and reliable model training process, PPO-Clip uses clip loss:

$$L^{clip}(\theta_A) = E[min(\rho_t(\theta_A)\hat{A}_t, clip(\rho_t(\theta_A), 1-\epsilon, 1+\epsilon))\hat{A}_t], \tag{22}$$

where ϵ is the clip range. Similar to the Policy Gradient algorithm [29], $\hat{A}_t$ denotes the estimated advantage function by $\pi_{\theta_A^{old}}$, denoted as:

$$\hat{A}_t = r_t + \gamma V_{\theta_C^{old}}(s_{t+1}) - V_{\theta_C^{old}}(s_t), \tag{23}$$

where γ denotes the discount factor and θ_C^{old} is a parameter of the old Critic network. Then, the model can perform gradient updates by sampling mini-batch data from the experience pool. The Actor network parameters θ_A can be updated as:

$$\theta_A = \theta_A^{old} - \alpha_A \nabla_{\theta_A} L^{clip}(\theta_A), \tag{24}$$

α_A is the learning rate of the Actor-network. The loss function of the Critic network is the mean square error:

$$L^C(\theta_C) = (V_{\theta_C}(s_t) - V_{tar}(s_t))^2, \tag{25}$$

where $V_{tar}(s_t)$ denotes the time difference error calculated from:

$$V_{tar}(s_t) = r_t + \gamma V_{\theta_C^{old}}(s_{t+1}). \tag{26}$$

Using α_C to denote the learning rate of the Critic network, the parameter update of the Critic network θ_C can be expressed as:

$$\theta_C = \theta_C^{old} - \alpha_C \nabla_{\theta_C} L^C(\theta_A). \tag{27}$$

5. Experimental evaluation

In this section, we verify the performance of the algorithms proposed in this paper by simulation. First, we introduce the experiment settings and the comparison algorithms. Then, to demonstrate the superiority of AQESF, we compared it to other algorithms in different aspects.

5.1. Experimental setup

Based on the problem modeling described in the previous sections, we use the deep learning framework PyTorch to achieve DRL-Queue and Multi-FM modules based on the multi-action PPO. The Actor and Critic networks of PPO comprise two multi-layer perception neural networks with 256 hidden neurons in each layer, ReLU as the activation function, and gradient updating using the Adam optimizer [30]. In order to obtain good training results, the system states are scaled to the same magnitude to eliminate the differences in data. More network parameters can be found in Table 2.

Without loss of generality, the dataset we use is similar to the setup of the study [6] with Alibaba trace for evaluation. Specifically, we scaled the tasks MI to less than 1000 MI based on the machine utilization. We generate arrival times for each BoT that follow a Poisson distribution. The memory requirements of tasks are sampled integers from a random distribution. We provide four types of VM instances, three each, to reflect different computational resources. Table 3 shows other parameters

Table 2
Parameters of the PPO-Clip model.

Parameter	Values
Training steps N	$5000 * n$
Discount factor γ	0.99
Learning rate $\alpha_A = \alpha_C$	3e−4
Mini-batch size	64
Clip coefficient ϵ	0.2
Replay Buffer Capacity C	2048
Update epochs K	5

Table 3
Parameters for simulation cloud environment.

Parameter	Values
Total number of VMs	12
Length of tasks	Convert utilization to tasks MI
Memory of tasks	Random Integer (0~1000) MB
Expected completion time	1.8s
Memory of VMs	{256,512,768,1024} MB
Frequency of VMs	{250,500,750,1000} MIPS

of the simulation environment. In addition, we set all the $task^{ect} = 1.8s$ to make sure that SLA is satisfied, i.e. $task^{ect} > \{VM^{proc}\}_{fast}/\{task^{len}\}_{\max}$.

We compare AQESF with existing algorithms for validation. The FIFO-manner and COP-manner algorithms are included:

- *MCT*: Select the VM with the shortest response time for the current task to schedule.
- *OLB*: Select the VM subset with the minimum idle time, then select the fastest process speed VM from those.
- *Min-Min*: Calculate the RT of each task in all available VMs, then select the task with the smallest RT from the BoT and assign it to the corresponding VM, and iterate through the BoT.
- *FGA* [18]: The FGA employs the Strengthen Elitist GA to deal with Bi-objectives COP. For better performance, we are consistent with FGA that uses Bi-objective weighting to design the fitness function F, i.e., Eq. (28). We limit the population size to 50 and the number of iterations to 200 to reduce the solving time. We adopt the experimental results with the best parameter $\omega \in (0 \sim 1)$:

$$F = \omega * \sum_{bs} \frac{task_j^{len}}{RT(task_j) * VM_i^{proc}} + (1-\omega) * \sum_{bs} task_j^{sr}. \tag{28}$$

- *CMA-DQN* [22]: We use the reward function proposed in Ref. [21] to obtain better performance of ART and SR, as shown below:

$$r_j = \begin{cases} \frac{task_j^{len}}{RT(task_j) * VM_i^{proc}}, & RT(task_j) > task^{ect} \\ 0, & RT(task_j) > task^{ect}. \end{cases} \tag{29}$$

- *RLQ* [7]: Based on Double DQN, the contextual bandit historical information is fully utilized as prior knowledge for model training. The reward function is also Eq. (29).

We answer the following questions about stability and superiority of AQESF through our experiments: ***Q1.*** How much is the performance improvement of our approach compared to different types of algorithms? ***Q2.*** How does our scheduling model perform in terms of cumulative performance? ***Q3.*** How does AQESF perform robustly compared to the widely adopted DRL scheduling algorithm? ***Q4.*** How much do DRL-Queue and Mutil-FM modules enhance the scheduling model?

5.2. EXP1: scheduling performance comparisons

To answer ***Q1***, we compare different algorithms on ART and SR with different distribution datasets and different average batch sizes (AvgBs) of tasks. The datasets are divided according to different ranges of tasks

Table 4
Performance comparison for 500 tasks in different datasets (Small, Medium and Large).

Small								
Metrics	AvgBs	MCT	OLB	Min-Min	FGA	CMA-DQN	RLQ	**AQESF**
ART	6	0.391	0.394	0.389	0.474	0.403	0.411	**0.387**
	8	0.452	**0.44**	0.44	0.507	0.456	0.472	0.445
	10	0.629	0.587	0.629	0.665	0.608	0.629	**0.574**
	12	0.912	0.875	0.925	0.961	0.907	0.912	**0.862**
SR	6	0.998	0.998	0.998	0.998	0.998	0.998	**0.998**
	8	0.998	0.998	0.998	0.998	0.998	0.998	**0.998**
	10	0.998	0.996	0.994	0.996	0.988	0.988	**0.998**
	12	0.856	0.88	0.86	**0.902**	0.866	0.876	0.9
Medium								
Metrics	AvgBs	MCT	OLB	Min-Min	FGA	CMA-DQN	RLQ	**AQESF**
ART	6	0.325	0.326	0.321	0.395	0.353	0.359	**0.32**
	8	0.602	0.57	0.586	0.614	0.598	0.607	**0.551**
	10	0.679	0.63	0.662	0.708	0.656	0.695	**0.587**
	12	0.908	0.84	0.883	0.883	0.91	0.92	**0.823**
SR	6	0.998	0.986	0.998	0.998	0.998	0.998	**0.998**
	8	0.922	0.96	0.914	0.958	0.946	0.93	**0.984**
	10	0.918	0.936	0.928	0.936	0.936	0.924	**0.972**
	12	0.852	0.89	0.862	0.896	0.878	0.872	**0.936**
Large								
Metrics	AvgBs	MCT	OLB	Min-Min	FGA	CMA-DQN	RLQ	**AQESF**
ART	6	0.482	0.474	0.487	0.56	0.502	0.5	**0.47**
	8	0.594	0.571	0.613	0.64	0.599	0.612	**0.563**
	10	0.793	0.775	0.807	0.855	0.811	0.825	**0.764**
	12	1.569	1.5	1.544	1.609	1.521	1.566	**1.468**
SR	6	0.994	0.998	0.99	0.992	0.992	0.994	**0.998**
	8	0.996	0.996	0.994	0.99	0.99	0.992	**0.998**
	10	0.868	0.862	0.866	**0.898**	0.864	0.882	0.886
	12	0.613	0.635	0.635	0.665	0.635	0.651	**0.679**

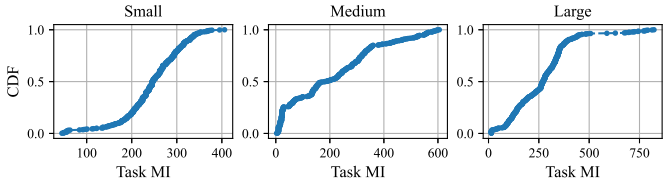


Fig. 6. Three different distributions of task MI.

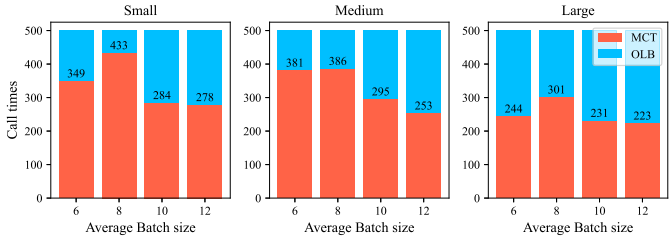


Fig. 7. Number of calls for MCT and OLB.

MI (Small, Medium and Large) and they have different CDF distributions, as shown in Fig. 6. The experimental results are shown in Table 4.

For the comparison of traditional heuristics (MCT, OLB and Min-Min), the clear trend is that as task MI or AvgBs increase, OLB is progressively superior to others in most cases. A crucial reason is that OLB optimizes VM utilization and improves the use of high-performance VMs. This is crucial in terms of considering long-term benefits. Among the algorithms only optimized for ART, the Min-Min is based on COP-manner and MCT is FIFO-manner, they do not show a clear pattern across datasets and AvgBs. This suggests that the short task prioritization brought about by Min-Min is not effective in improving ART and that the task ordering needs to be adaptive. In addition, compared to the methods that also use COP-manner for scheduling, FGA is optimized for bi-objective and shows significant improvement in SR compared to Min-Min. However, compared to Min-Min, which is optimized for ART only, FGA still performs poorly in ART, even with the best parameter ω chosen in the fitness function.

When comparing algorithms incorporating DRL techniques, it is clear that the Bi-objectives optimization maintains: AQESF > CMA-DQN > RLQ. The critical reason is that AQESF is based on the COP-manner, utilizing a batch of task information to assist in scheduling decisions. In contrast, both the CMA-DQN and the RLQ operate in a FIFO-manner, neglecting the performance impact caused by task ordering.

The AQESF leverages Multi-FM to confine the search space to a more optimal domain, coupling with DRL-Queue for cumulative performance enhancement, which ultimately leads to the SOTA performance for optimization with Bi-objectives. However, we still find that AQESF is still sub-optimal in some results, such as the case (Small, AvgBs = 8, ART), where ART is weak, which occurs at SR = 0.998, it is difficult to improve ART because the reward has converged to almost optimal and the reward function is designed for SR only. For the case of weak SR (Small, AvgBs = 12, SR; Large, AvgBs = 10, SR), this is a problem caused by the lack of search space due to task placement with only two algorithms to execute.

In addition, we are very interested in the number of calls for the two algorithms in the Multi-FM, so we show it in Fig. 7. We can find that the OLB increases gradually as the AvgBs increase. Except for AvgBs = 6, this is also due to the fact that the reward is almost optimal and the reward function is designed for SR only, resulting in Multi-FM not being able to learn further.

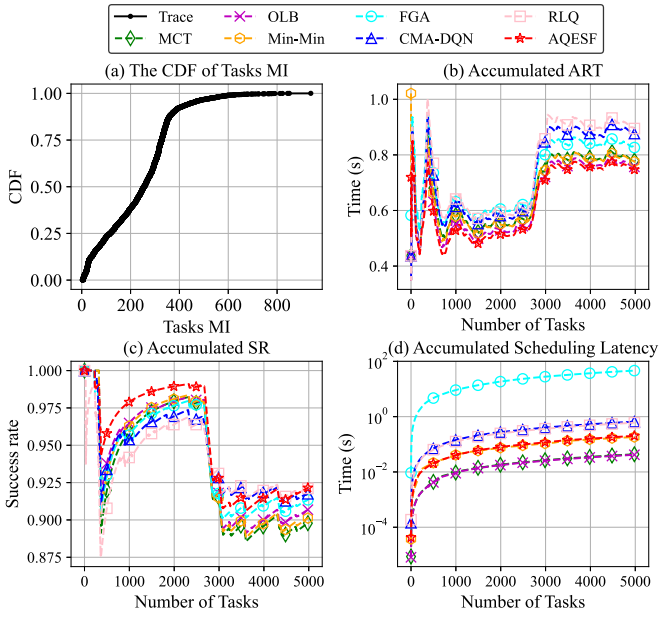


Fig. 8. Cumulative performance of different scheduling algorithms with AvgBs = 10, 5000 tasks.

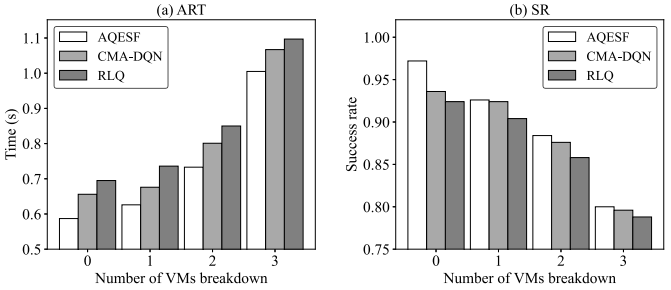


Fig. 9. Robustness performance of DRL integrating algorithms in VMs breakdown with AvgBs = 10 (Medium).

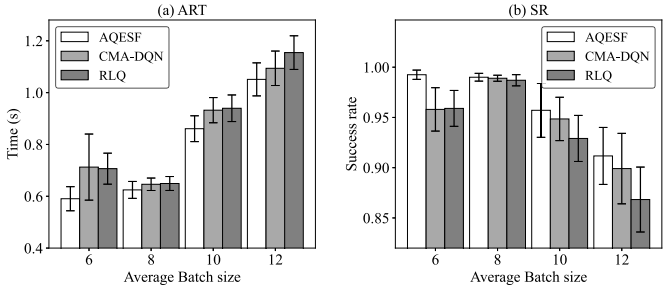


Fig. 10. Robustness performance of DRL integrating algorithms in Random benchmark with different AvgBs (Medium).

5.3. EXP2: scalability evaluation

Concerning ***Q2***, we investigate the cumulative performance of the scheduling model over more prolonged traces, as shown in Fig. 8. For the cumulative ART and SR, it can be found that with the growing task sequences and different task request fluctuations, our method always shows optimal performance in terms of ART. Moreover, the performance in SR is better than that of CMA-DQN and the RLQ in most cases. Compared with other scheduling algorithms not supported by the DRL technique, AQESF has 3.7 %~16.6 % latency reduction in ART and 0.1 %~2.7 % improvement in SR. In addition, starting from $task \approx 3000$, the task size becomes larger, and although the traditional heuristic algorithm (MCT, OLB and Min-Min) optimized for ART still maintains a better ART, there is a serious decline in SR. In contrast, the algorithm (FGA, CMA-DQN and RLQ) optimized for SR still maintains high SR.

For the cumulative latency of different algorithms for task scheduling decisions, the FGA algorithm exhibits high scheduling latency and cannot be applied to latency-sensitive online scheduling decisions. The other algorithms are all able to process 5000 task decisions in less than $1s$. The scheduling latency of AQESF is similar to COP-manner traditional heuristic algorithms with time complexity $O(bs^2)$. It is better than the CMA-DQN and the RLQ since AQESF can make decisions for a BoT by inferring the DRL model only once. Whereas the CMA-DQN and the RLQ need to go through the DRL model bs times to generate inferred decisions, i.e., the time complexity of both the CMA-DQN and the RLQ is $O(\kappa \times bs)$. The rest of the traditional heuristic algorithms have shorter scheduling latency, which is predictable. Table 5 shows more detailed inference latency of different algorithms, and our proposed AQESF and Min-Min are on the same order of magnitude.

Table 5
Average inference latency per task (ms).

MCT	OLB	Min-Min	FGA	CMA-DQN	RLQ	AQESF
0.009	0.008	0.037	9.273	0.132	0.121	0.041

5.4. EXP3: scheduling robustness comparison

In this subsection, the ***Q3*** will be answered. Considering the data center environment, unpredictable accidents, such as server downtime leading to unavailability of nodes or drift in the distribution of workload, are possible. In this set of experiments, we compare the robustness performance of algorithms with DRL integration.

First, we compare the performance of algorithms in processing 500 tasks when the number of VM breakdowns is 1,2,3, respectively. From the experimental results in Fig. 9, it can be found that all scheduling models undergo different degrees of degradation in both ART and SR as the number of VMs breakdown increases. The Bi-objectives performance always maintains: AQESF > CMA-DQN > RLQ. Compared to CMA-DQN, AQESF maintains a 5.8 % to 8.5 % time overhead reduction in ART and 0.2 % to 0.9 % improvement in SR.

Moreover, these three algorithms are integrated with the DRL technique, which is trained by Alibaba trace, and then these scheduling models are applied to the Random benchmark. We conducted ten separate experiments to test the performance in the face of data distribution drift over a short period of time (200 tasks in 20s), as shown in Fig. 10. It can be observed that the average performance of the CMA-DQN and the RLQ models is always weaker than that of AQESF under different AvgBs, and the standard deviation is also relatively poor. In the worst case of CMA-DQN, AQESF has a 17.3 % reduction in ART and a 3.5 % increase in SR. Predictably, if both task scheduling order and placement decisions are learned and inferred using DRL, it may lead to poor model robustness, which is not friendly for real-world scenarios.

5.5. EXP4: ablation study

Next, for ***Q4***, we conducted ablation experiments to demonstrate the enhancement in scheduling performance through the joint optimization of DRL-Queue and Multi-FM in AQESF. For model training and comparison, we remove the DRL-Queue (MCT + OLB), the OLB (DRL + MCT), and the MCT (DRL + OLB), respectively. The MCT + OLB indicates that the Multi-FM will decide which scheduling algorithm to use for each task placement. Table 6 shows that our method remains optimal on both ART and SR. This suggests that AQESF is able to find better solutions while

Table 6
Comparative results of ablation studies (Medium).

Metrics	Algorithm	AvgBs			
		6	8	10	12
ART	**AQESF**	**0.32**	**0.551**	**0.587**	**0.823**
	DRL + MCT	0.327	0.575	0.63	0.872
	DRL + OLB	0.33	0.555	0.587	0.824
	MCT + OLB	0.327	0.568	0.624	0.842
SR	**AQESF**	**0.998**	**0.984**	**0.972**	**0.936**
	DRL + MCT	0.998	0.956	0.95	0.896
	DRL + OLB	0.986	0.978	0.968	0.932
	MCT + OLB	0.998	0.97	0.936	0.9

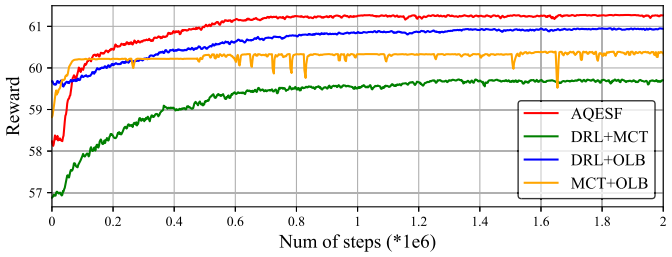


Fig. 11. Convergence curves for comparison models in ablation experiments (Medium, AvgBs = 8).

covering a wider optimization space. In addition, DRL-Queue does improve the performance of the greedy algorithm (Compared to the MCT and OLB results in Table 4), but the DRL + single greedy algorithm really does not handle dynamic user requests efficiently (As shown in Example 1).

From the model convergence curve in Fig. 11, it is found that our model gradually converges to the optimum, which implies that AQESF can organically combine the DRL-Queue and the Multi-FM to realize the approximation of optimal solutions. Overall, the performance of ART and SR maintains: AQESF > DRL + OLB > MCT + OLB > DRL + MCT.

6. Potential applications

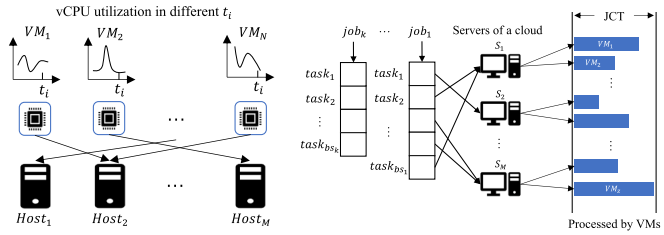
Potential applications of AQESF will be discussed in this section. Specifically, the problem addressed by AQESF is an online scheduling problem for a batch of workloads. We elaborate on potential applications of AQESF to the following scenarios: VM placement [8] and computation offloading [31].

VM Placement. In VM consolidation, VM placement is regarded as a bin packing problem. Moreover, the requests within VMs persistently arrive and execute, leading to fluctuations in vCPU utilization of VMs, as depicted in Fig. 12(a). This means that the placement decision for that batch of VMs at time t_i has a lasting impact on subsequent system changes. The problem is modeled similarly as described in Fig. 3.

Researchers often model the current batch of VMs to be placed, together with the servers, as COP, and use traditional heuristics [16] or meta-heuristics to solve it [32], and do not consider the cumulative performance. Alternatively, the batch of VMs is placed one by one in the default order using DRL [8], which cannot fully utilize the information of the current batch of VMs. None of the existing work can take into account both sequential decision-making and COP solving.

Computation Offloading. In computation offloading, as shown in Fig. 12(b). The compute nodes have different capabilities and resource constraints, and the user offloads the BoT within the job to these nodes for processing. The maximum response time of the BoT is the job completion time (JCT), which can be reduced to a NP-hard partitioning problem [31].

Many studies typically formulate this offloading problem as COP with the primary goal of minimizing JCT. However, they do not consider the impact of task placement on subsequent system states. The AQESF can be applied to optimize the cumulative rewards by further enhancing the existing state-of-the-art FIFO-manner approaches.



(a) VM placement scenario. (b) Computation Offloading scenario.

Fig. 12. Potential Applications of AQESF.

7. Conclusion and future work

In this study, our focus lies in addressing the generalized online BoT scheduling problem while optimizing cumulative performance. We model this problem as a sequential decision problem incorporating COP. Further, we formulate the online BoT scheduling problem as an MDP and AQESF to solve it. Experiments demonstrate that AQESF achieves SOTA results in both ART and SR compared to traditional heuristics, meta-heuristics, and DRL-based methods. Moreover, our method exhibits stronger robustness and lower scheduling latency compared to the widely used DRL task scheduling paradigm.

In future work, we discuss potential scenarios of AQESF. The scheduling framework fits well with the widely studied problems of VM placement and computation offloading. This means we can use the AQESF framework to further enhance the performance of the FIFO-manner algorithms that have performed well within these problems. Additionally, we can explore the impact of weighted reward functions on the optimization objective within this framework, as well as conduct simultaneous exploration of both scheduling order and placement using DRL to discover optimal solutions.

CRediT authorship contribution statement

Huikang Huang: Writing – review & editing, Writing – original draft, Validation, Methodology, Formal analysis, Data curation, Conceptualization; **Weiwei Lin:** Writing – review & editing, Supervision, Funding acquisition; **Minxian Xu:** Writing – review & editing; **Keqin Li:** Writing – review & editing.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by Guangzhou Development Zone Science and Technology Project (2023GH02), Guangdong Provincial Natural Science Foundation Project (2025A1515010113), Shandong Provincial Natural Science Foundation Project (ZR2024LZH012) and the Major Key Project of PCL, China under Grant PCL2025AS11.

References

[1] C. Wu, R. Buyya, K. Ramamohanarao, Cloud pricing models: taxonomy, survey, and interdisciplinary challenges, ACM Comput. Surv. 52 (6) (2019). https://doi.org/10.1145/3342103

[2] A. Alashaikh, E. Alanazi, A. Al-Fuqaha, A survey on the use of preferences for virtual machine placement in cloud data centers, ACM Comput. Surv. 54 (5) (2021). https://doi.org/10.1145/3450517

[3] H. Yuan, J. Bi, M. Zhou, Q. Liu, A.C. Ammari, Biobjective task scheduling for distributed green data centers, IEEE Trans. Autom. Sci. Eng. 18 (2) (2021) 731–742. https://doi.org/10.1109/TASE.2019.2958979

[4] Y. Yang, H. Shen, Deep reinforcement learning enhanced greedy optimization for online scheduling of batched tasks in cloud HPC systems, IEEE Trans. Parallel Distrib. Syst. 33 (11) (2022) 3003–3014. https://doi.org/10.1109/TPDS.2021.3138459

[5] Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, IEEE Trans. Parallel Distrib. Syst. 11 (7) (2000) 760–768. https://doi.org/10.1109/71.877834

[6] Z. Tong, F. Ye, B. Liu, J. Cai, J. Mei, DDQN-TS: A novel bi-objective intelligent scheduling algorithm in the cloud environment, Neurocomputing 455 (2021) 419–430. https://doi.org/10.1016/j.neucom.2021.05.070

[7] A. Staffolani, V.-A. Darvariu, P. Bellavista, M. Musolesi, RLQ: workload allocation with reinforcement learning in distributed queues, IEEE Trans. Parallel Distrib. Syst. 34 (3) (2023) 856–868. https://doi.org/10.1109/TPDS.2022.3231981

[8] J. Zeng, D. Ding, K. Kang, H. Xie, Q. Yin, Adaptive DRL-based virtual machine consolidation in energy-efficient cloud data center, IEEE Trans. Parallel Distrib. Syst. 33 (11) (2022) 2991–3002. https://doi.org/10.1109/TPDS.2022.3147851

[9] Z. Zhang, M. Zhao, H. Wang, Z. Cui, W. Zhang, An efficient interval many-objective evolutionary algorithm for cloud task scheduling problem under uncertainty, Inf. Sci. 583 (2022) 56–72. https://doi.org/10.1016/j.ins.2021.11.027

[10] F. Shi, A genetic algorithm-based virtual machine scheduling algorithm for energy-efficient resource management in cloud computing, Concurr. Comput. 36 (22) (2024) e8207. https://doi.org/10.1002/cpe.8207

[11] S. Bhetiwal, S.K. Misra, Survey on task scheduling with ant colony optimization, in: 2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC), 2023, pp. 690–696. https://doi.org/10.1109/ICSCCC58608.2023.10176927

[12] R. Mohamed, M. Avgeris, A. Leivadeas, I. Lambadaris, Optimizing resource fragmentation in virtual network function placement using deep reinforcement learning, IEEE Trans. Mach. Learn. Commun. Netw. 2 (2024) 1475–1491. https://doi.org/10.1109/TMLCN.2024.3469131

[13] G. Gupta, N. Mangla, An enhanced MIN-MIN algorithm for workflow scheduling in cloud computing, in: 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), 2022, pp. 2084–2091. https://doi.org/10.1109/ICAC3N56670.2022.10074016

[14] L. Wang, E. Gelenbe, Adaptive dispatching of tasks in the cloud, IEEE Trans. Cloud Comput. 6 (1) (2018) 33–45. https://doi.org/10.1109/TCC.2015.2474406

[15] B. Zheng, L. Pan, S. Liu, Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments, J. Syst. Softw. 176 (2021) 110934. https://doi.org/10.1016/j.jss.2021.110934

[16] W. Lin, W. Wu, L. He, An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers, IEEE Trans. Serv. Comput. 15 (2) (2022) 766–777. https://doi.org/10.1109/TSC.2019.2961082

[17] M. Kumar, S.C. Sharma, PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing, Neural Comput. Appl. 32 (2020) 12103–12126.

[18] H. Sun, B. Zhang, X. Zhang, Y. Yu, K. Sha, W. Shi, FlexEdge: dynamic task scheduling for a UAV-based on-demand mobile edge server, IEEE Internet Things J. 9 (17) (2022) 15983–16005. https://doi.org/10.1109/JIOT.2022.3152447

[19] S. Mousavi, S.E. Mood, A. Souri, M.M. Javidi, Directed search: a new operator in NSGA-II for task scheduling in IoT based on cloud-fog computing, IEEE Trans. Cloud Comput. 11 (2) (2023) 2144–2157. https://doi.org/10.1109/TCC.2022.3188926

[20] U. Demirbaga, EcoCloud:Green computing through energy and carbon efficient task scheduling in industrial IoT-enabled cloud environments, IEEE Internet Things J. (2025) 1. https://doi.org/10.1109/JIOT.2025.3537111

[21] Y. Huang, L. Cheng, L. Xue, C. Liu, Y. Li, J. Li, T. Ward, Deep adversarial imitation reinforcement learning for QoS-aware cloud job scheduling, IEEE Syst. J. 16 (3) (2022) 4232–4242. https://doi.org/10.1109/JSYST.2021.3122126

[22] X. Tang, F. Liu, B. Wang, M. Zhang, J. Jiang, Q. Tang, Q. Wu, C.L.P. Chen, Cost and makespan-aware task scheduling with deep reinforcement learning in multicloud environments, IEEE Trans. Comput. Social Syst. (2025) 1–14. https://doi.org/10.1109/TCSS.2025.3553856

[23] D. Cui, Z. Peng, J. Xiong, B. Xu, W. Lin, A reinforcement learning-Based mixed job scheduler scheme for grid or IaaS cloud, IEEE Trans. Cloud Comput. 8 (4) (2020) 1030–1039. https://doi.org/10.1109/TCC.2017.2773078

[24] L. Cheng, A. Kalapgar, A. Jain, Y. Wang, Y. Qin, Y. Li, C. Liu, Cost-aware real-time job scheduling for hybrid cloud using deep reinforcement learning, Neural Comput. Appl. 34 (21) (2022) 18579–18593.

[25] R. Buyya, R. Ranjan, R.N. Calheiros, Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities, in: 2009 International Conference on High Performance Computing & Simulation, 2009, pp. 1–11. https://doi.org/10.1109/HPCSIM.2009.5192685

[26] S. Martello, P. Toth, Solution of the zero-one multiple knapsack problem, Eur. J. Oper. Res. 4 (4) (1980) 276–283. Combinational Optimization. https://doi.org/10.1016/0377-2217(80)90112-5

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, 2017, arXiv:1707.06347

[28] S. Huang, S. Ontañón, A closer look at invalid action masking in policy gradient algorithms, The International FLAIRS Conference Proceedings 35 (2022). https://doi.org/10.32473/flairs.v35i.130584

[29] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: M.F. Balcan, K.Q. Weinberger (Eds.), Proceedings of The 33rd International Conference on Machine Learning, 48 PMLR, New York, New York, USA, 2016, pp. 1928–1937.

[30] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, 2017, arXiv:1412.6980

[31] K. Li, Heuristic computation offloading algorithms for mobile users in fog computing, ACM Trans. Embed. Comput. Syst. 20 (2) (2021). https://doi.org/10.1145/3426852

[32] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, Q. Zheng, Power-aware and performance-guaranteed virtual machine placement in the cloud, IEEE Trans. Parallel Distrib. Syst. 29 (6) (2018) 1385–1400. https://doi.org/10.1109/TPDS.2018.2794369

Huikang Huang received the M.S. degree in Computer Science and Theory from the South China Agricultural University, Guangzhou, China, in 2021. Now, he is a Ph.D. candidate in the School of Computer Science and Engineering, South China University of Technology. His research interests mainly focus on cloud computing and reinforcement learning.

Weiwei Lin (Senior Member, IEEE) received the B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and the Ph.D. in Computer Application from South China University of Technology in 2007. Currently, he is a professor in the School of Computer Science and Engineering at South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He has published more than 200 papers in refereed journals and conference proceedings. He has been a reviewer for many international journals, including ICML, IEEE TPDS, TSC, TCC, TC, TCYB, etc. He is a distinguished member of CCF and a senior member of the IEEE.

Minxian Xu is currently an associate professor at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. He received the BSc degree in 2012 and the MSc degree in 2015, both in software engineering from the University of Electronic Science and Technology of China. He obtained his PhD degree from the University of Melbourne in 2019. His research interests include resource scheduling and optimization in cloud computing. He has coauthored 50+ peer-reviewed papers published in prominent international journals and conferences, such as ACM Computing Surveys, IEEE Transactions on Sustainable Computing, IEEE Transactions on Automation Science and Engineering, Journal of Parallel and Distributed Computing, Journal of Systems and Software and International Conference on Service-Oriented Computing. His Ph.D. The thesis was awarded the 2019 IEEE TCSC Outstanding Ph.D. Dissertation Award. He is a member of IEEE and CCF. More information can be found at: minxianxu.info.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published over 740 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He has served on the editorial boards of the IEEE TPDS, TC, TCC, TSC, and TSUSC. He is an IEEE Fellow.