Cost-aware cloud workflow scheduling using DRL and simulated annealing

Gu, Y., Cheng, F., Yang, L., Xu, J. ORCID: https://orcid.org/0009-0009-2964-8971, Chen, X. ORCID: https://orcid.org/0000-0001-9267-355X and Cheng, L. ORCID: https://orcid.org/0000-0003-1638-059X (2024) Cost-aware cloud workflow scheduling using DRL and simulated annealing. Digital Communications and Networks, 10 (6). pp. 1590-1599. ISSN 2352-8648 doi: 10.1016/j.dcan.2023.12.009 Available at https://centaur.reading.ac.uk/115740/

Digital Communications and Networks 10 (2024) 1590–1599

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Cost-aware cloud workflow scheduling using DRL and simulated annealing

Yan Gu [a], Feng Cheng [b,*], Lijie Yang [a], Junhui Xu [a], Xiaomin Chen [c], Long Cheng [a]

[a] School of Control and Computer Engineering, North China Electric Power University, Beijing 100026, China
[b] School of Mathematics, Southwest Jiaotong University, Chengdu 610032, China
[c] Department of Computer Science, University of Reading, Reading RG6 6AH, UK

ARTICLE INFO

Keywords:
Cloud computing
Deep reinforcement learning
Simulated annealing algorithm
Job scheduling
Workflow

ABSTRACT

Cloud workloads are highly dynamic and complex, making task scheduling in cloud computing a challenging problem. While several scheduling algorithms have been proposed in recent years, they are mainly designed to handle batch tasks and not well-suited for real-time workloads. To address this issue, researchers have started exploring the use of Deep Reinforcement Learning (DRL). However, the existing models are limited in handling independent tasks and cannot process workflows, which are prevalent in cloud computing and consist of related subtasks. In this paper, we propose SA-DQN, a scheduling approach specifically designed for real-time cloud workflows. Our approach seamlessly integrates the Simulated Annealing (SA) algorithm and Deep Q-Network (DQN) algorithm. The SA algorithm is employed to determine an optimal execution order of subtasks in a cloud server, serving as a crucial feature of the task for the neural network to learn. We provide a detailed design of our approach and show that SA-DQN outperforms existing algorithms in terms of handling real-time cloud workflows through experimental results.

1. Introduction

Cloud computing has revolutionized the delivery of computing services over the Internet. Its flexibility, scalability, and high performance have led to rapid development in the field of information technology and made it one of the most promising techniques in today's business world. Infrastructure-as-a-Service (IaaS) is a cloud service model that provides businesses with powerful computing and storage resources, allowing enterprises to deploy and run their online services in a convenient and cost-efficient manner [1]. As a result, more and more companies are utilizing cloud computing as their service platforms, leading to the development of increasingly complex cloud-based services [2].

Effective task scheduling strategies are critical in cloud computing to efficiently dispatch computing tasks to a resource pool of Virtual Machines (VMs), given that scheduling directly affects service performance and operational costs [3]. Despite significant attention from both academia and industry in recent years, most cloud task scheduling approaches focus on handling batch tasks rather than real-time tasks [4], such as transactional workloads [5] that are common in various domains such as e-commerce, with no fixed task arrival patterns. Service providers may experience performance degradation due to the instability of such workloads [6]. As cloud computing is market-oriented, advanced task scheduling strategies that can effectively handle such real-time tasks are required to maximize resource utilization, reduce costs, and improve service quality.

Workflows are a prevalent computational model extensively used in various scientific domains and applications [7]. Workflows are typically modeled as Directed Acyclic Graphs (DAGs) and can be broken down into a set of related subtasks. Consequently, the cloud workflow scheduling problem is more intricate than general task scheduling and remains a significant challenge. Unlike independent tasks, which are typically isolated, subtasks within a workflow are dependent on each other, making it difficult to optimize resource utilization and reduce costs while maintaining high performance. Workflow scheduling, as an NP-hard problem, has consistently been a crucial research topic in the field. In cloud computing, workflow scheduling refers to the process of entailing the allocation of tasks to suitable VMs and executing them. The objective of this paper is to investigate a highly efficient real-time workflow scheduling framework capable of adapting to diverse workloads.

Numerous advanced workflow scheduling methods have been proposed in the literature to handle the challenges of workflow scheduling in cloud computing. Metaheuristic algorithms are a typical approach that is easy to implement, presents low computational complexity,

* Corresponding author.
E-mail address: chengfeng2013@swjtu.edu.cn (F. Cheng).

https://doi.org/10.1016/j.dcan.2023.12.009
Received 31 March 2023; Received in revised form 5 December 2023; Accepted 27 December 2023
Available online 2 January 2024

and can provide near-optimal solutions, making them widely used in workflow scheduling. For instance, Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and other nature-inspired methods have been utilized for workflow scheduling [8]. However, most of the existing approaches are designed for specific workloads or metrics, such as satisfying user-defined deadlines with minimal costs [9] or minimizing makespans [10]. Few methods have considered real-time cases, in which workflows could be submitted by different users at any time, and users may have unique service requirements. This makes real-time workflow scheduling in cloud computing a challenging problem, requiring the development of new approaches that can handle the complexity and dynamic nature of real-time workflows.

As mentioned above, the challenges in real-time workflow scheduling encompass more than just considering task dependencies, resource constraints, and uncertainties to enhance workflow performance and reliability. It is equally important to account for real-time requirements, such as the unique service demands of users. To address the challenges of real-time workflow scheduling in cloud computing, we propose a novel intelligent cloud workflow scheduling framework called SA-Deep Q-Network (SA-DQN). The framework is based on the seamless integration of SA [11] and Deep Reinforcement Learning (DRL) [12]. SA-DQN can learn effective scheduling strategies to optimize both the Quality of Service (QoS) and monetary costs on VMs in an automatic way. DRL is a popular machine learning technique that allows an agent to learn how to solve complex control problems by interacting with systems. We leverage DRL to handle the real-time workflows in cloud computing. Furthermore, the SA algorithm is utilized to obtain an optimal execution order of sub-tasks in a cloud server. This execution order is used as a key characteristic of the task so that the DRL agent can learn and optimize the scheduling strategy effectively. By combining SA and DRL, our proposed framework can address the complexity and dynamic nature of real-time workflow scheduling in cloud computing, while maximizing resource utilization and minimizing operational costs.

In general, the main contributions of this paper can be summarized as follows:

- We propose a DRL-based scheduling algorithm to address the real-time scheduling problem of concurrent workflow-type tasks.
- We integrate metaheuristics with DRL-based scheduling in a seamless way, and use the SA algorithm as a case to extract the features of tasks and cloud servers for facilitating the learning steps in DRL-based algorithm.
- We compare our method with some other real-time scheduling algorithms, and the experimental results show that our method outperforms the other algorithms on costs and job completion time under different workflow loads and VM configurations.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the system model and the workflow scheduling problem for the cloud-based application. Section 4 presents the implementation of our proposed SA-DQN algorithm. Section 5 reports the performance evaluation for our approach, and Section 6 concludes this paper with remarks on our future work.

2. Related work

Task scheduling is widely recognized as a critical and challenging problem in various domains like IoT, edge, and cloud computing [23,24]. Over the past few years, numerous advanced algorithms have been proposed to tackle cloud task scheduling. For instance, [15] introduces an efficient search algorithm based on the Markov decision process. The authors in [25] propose a dynamic task scheduling algorithm employing a two-stage strategy. Additionally, [26] explores the application of the Whale Optimization Algorithm (WOA) in cloud task scheduling with multi-objective optimization. However, these algorithms primarily focus on scheduling independent tasks. As the complexity of applications increases, there is a growing research interest in workflow scheduling [27], aiming to enhance cloud service performance through relevant scheduling strategies. In this context, our paper contributes to the field by focusing on designing scheduling methods specifically tailored for workflows.

Workflows are commonly used to describe data-intensive applications, and scheduling them poses a challenging problem in the field of research. Task dependencies, resource constraints, and execution time are crucial factors considered during workflow scheduling. Numerous studies have explored the application of various heuristic algorithms to address these challenges. Notable heuristic algorithms employed for solving workflow scheduling problems include PSO [13], ACO [14], and Genetic Algorithms (GA) [15]. Additionally, in [16], a fuzzy-dominance-based Heterogeneous Earliest Finish Time (HEFT) algorithm is proposed. The authors in [28] introduce a hybrid heuristic algorithm based on enhanced task-type priorities for workflow scheduling in deadline-constrained cloud environments. In contrast to the aforementioned approaches that solely rely on heuristic algorithms, this paper combines heuristic algorithms and DRL to achieve efficient scheduling.

DRL has demonstrated its effectiveness in handling highly dynamic and complex environments, making it a popular choice for addressing intricate decision-making problems, such as vehicle control [29] and network optimization [30]. Given these advantages, an increasing number of researchers are adopting DRL as a scheduling strategy for workflows. For example, [17] proposes an algorithm that leverages DRL to enhance near-end policy optimization techniques, tackling a complex workflow scheduling problem in edge-cloud environments. The authors in [18] formulate the workflow scheduling problem as a multi-objective optimization challenge, introducing a design scheme based on DRL to address the dynamic and evolving system scheduling decisions. Additionally, [31] presents a two-step service offloading approach based on DRL, aiming to reduce the costs associated with edge servers. Furthermore, [19] proposes a DRL-based workflow scheduling method that considers changes in the network and machine states in the cloud, with the objective of minimizing the workflow makespan. However, most of the aforementioned scheduling algorithms do not specifically consider real-time workflows.

In recent years, the design of real-time workflow scheduling has garnered significant attention. For instance, [20] proposes an improved PSO-based algorithm for real-time workflow scheduling, enabling adaptive optimization of resource utilization. The authors in [8] present a real-time multi-workflow scheduling scheme that dynamically schedules workflows with minimal costs under different deadline constraints. Additionally, [21] develops a deadline-aware heuristic algorithm for scheduling workflows with uncertain task execution time and random arrivals. Furthermore, [22] introduces a dynamic workflow scheduling algorithm that strikes a balance between the costs, system resource utilization, and deviations.

Table 1 provides a summary of the key characteristics of the literature related to workflow scheduling. They are primarily divided into real-time workflows and batch workflows. In contrast to these approaches based solely on heuristic algorithms or DRL, this paper proposes an algorithm that combines heuristic algorithms and DRL to design a scheduling algorithm specifically tailored for real-time workflows. The SA algorithm determines the optimal execution order of sub-tasks within each workflow, consequently optimizing the overall workflow execution time. Subsequently, DRL is utilized to allocate the entire workflow to the most suitable VM.

3. System model

This section introduces cloud scheduling and the system model used in this study. To facilitate the presentation, we provide a list of the notations used in Table 2.

Table 1
Typical methods for compressing smart meter data in literature.

Reference	Job type	Scheduling method	Optimization objective
M. H. Shirvani et al. [13]	Batch workflow	Particle swarm optimization	Makespan
Y. H. Jia et al. [14]	Batch workflow	Ant colony optimization	Total cost and execution time
H. Y. Shishido et al. [15]	Batch workflow	Particle swarm and genetic optimization	Execution cost and execution time
X. Zhou et al. [16]	Batch workflow	Fuzzy dominance sort	Cost and makespan
A. Jayanetti et al. [17]	Batch workflow	Deep reinforcement learning	Makespan and energy consumption
R. Xie et al. [18]	Batch workflow	Deep reinforcement learning	Time and energy consumption
Y. Xiang et al. [19]	Batch workflow	Deep reinforcement learning	Makespan
P. Guo et al. [20]	Real-time workflow	Particle swarm optimization	Cost and makespan
X. Ma et al. [8]	Real-time workflow	Dynamic programming	Rental cost
J. Liu et al. [21]	Real-time workflow	Dynamic programming	Monetary cost
H. Chen et al. [22]	Real-time workflow	Dynamic programming	Cost and resource utilization

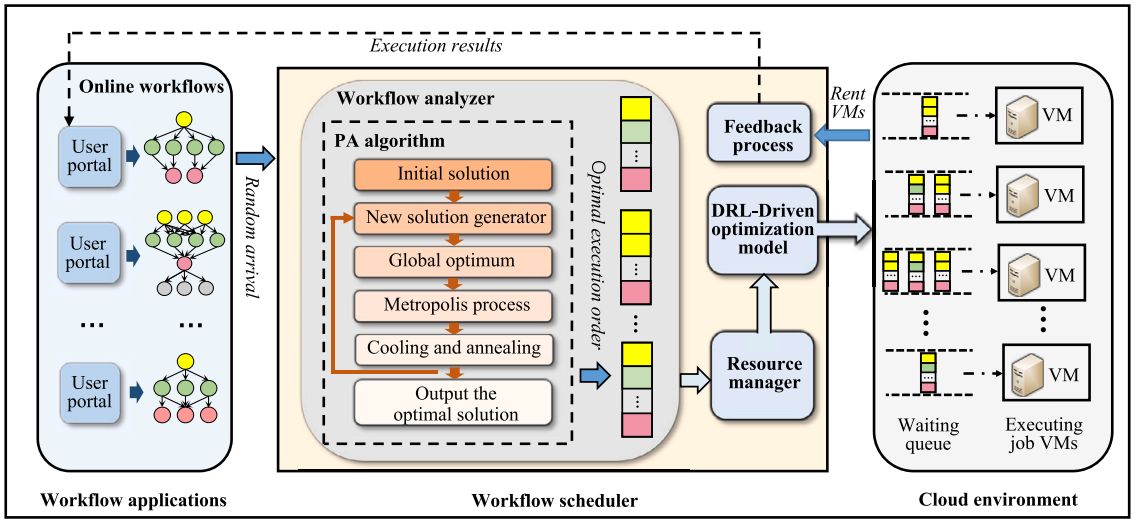


Fig. 1. The cloud workflow task scheduling system.

Table 2
The used notations.

Notation	Meaning
J_{ID}	ID of a job
J_{SE}	Subtask evaluation array
J_{SRM}	Subtask relationship matrix
QoS	Expected response time of users
VM_{ID}	ID of a cloud instance
VM_{COM}	Amount of computing in a cloud instance computing unit
VM_{NCU}	Number of cloud instance computing units
VM_{COST}	Costs of cloud instances
T_{rep}	Response time of a job
T_{exe}	Execution time of a job
T_{wait}	Wait time of a job
SJ_i	ID of the ith subtask
Δf	Cost function difference
p	Probability that SA receives a new solution

3.1. System architecture

To streamline the cloud environment, we categorize it into 3 layers: IaaS providers, application vendors, and end-users. In this setup, an application vendor rents a number of VM instances from an IaaS provider and deploys its service on the VMs. When a user submits a request to the online service, a complex job is generated. Upon the arrival of a workflow, a metaheuristic algorithm (i.e., SA) is used to obtain the optimal subtask execution order. The workflow is then sent to the queue of an assigned VM instance for execution. Without loss of generality, we assume that each VM contains a set of computing cores, and each core can handle only 1 subtask at a time. The role of the scheduler, as explored in this research, is to minimize costs while guaranteeing QoS [32].

The general system architecture for cloud workflow scheduling is illustrated in Fig. 1. When a job (i.e., workflow) arrives, the job information and current VM instances are fed into the feature extractor. The scheduler then sends the job to the queue of a specified instance based on the extracted characteristics and optimization objectives. Each job will have to wait until the previous job in the same queue is completed.

3.2. Scheduling model

3.2.1. Workflow characteristics

To execute a workflow, it is essential to understand the interdependencies between sub-tasks. For instance, some sub-tasks can only be executed after the completion of other specific sub-tasks, known as preconditions. By organizing subtasks into layers, we can formulate the structure of the subtasks. As an example, consider Fig. 2, where subtasks 2 and 3 depend on the execution of subtask 1. This allows us to obtain topological relationships among subtasks, which can be represented using matrices. Based on this, an arriving workflow can be represented as

$$Job_i = \{J_{ID}, J_{SE}, J_{SRM}, QoS\} \tag{1}$$

where J_{ID} is the job's identification, J_{SE} is an array that records the amount of required computation for each subtask, measured in points [26], J_{SRM} is the relation matrix of subtasks, and QoS is the expected response time from the user.

3.2.2. Cloud instances

In practical cloud computing scenarios, service providers rent cloud instances to deploy their applications. In the context of our study, we consider the cloud instance model as a fundamental building block of

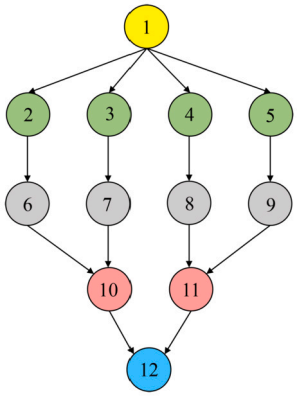


Fig. 2. An example of a workflow structure.

the cloud system. The computing power of an instance depends on the number and performance of its cores. We assume that a service provider has reserved a limited number of VM instances from the resource pool and utilizes them in an on-demand manner. For each instance, we provide the following definition.

$$VM_j = \{VM_{ID}, VM_{COM}, VM_{NCU}, VM_{COST}\} \tag{2}$$

where VM_{ID} denotes the ID of the cloud instance, VM_{COM} represents the computing capability (in points) for each computing unit or core of the cloud instance, VM_{NCU} denotes the number of cores, and VM_{COST} represents the usage cost of the cloud instance.

3.2.3. Workflow scheduling mechanism

The scheduler is responsible for allocating workflows to suitable VMs. When a workflow arrives, the SA algorithm is used to obtain the optimal sub-task execution order. The scheduler determines the instance to which the workflow should be assigned based on the time dimension feature obtained from the execution order. Once a workflow is assigned, the included subtasks are added to the task queue of the instance following the previously calculated optimal order.

Task response time T_{rep} can be defined as

$$T_{rep} = T_{exe} + T_{wait} \tag{3}$$

where T_{exe} is the execution time of the workflow obtained by the SA algorithm, and T_{wait} is the period from the arrival time point of the workflow to the beginning time point of the execution. The waiting time T_i^{wait} of a workflow can be calculated as

$$T_i^{wait} = \begin{cases} \sum_{i=0}^{s} T_i^{exe}, & \text{if } s > 0 \\ 0, & \text{if } s = 0 \end{cases} \tag{4}$$

where s is the location of the instance queue assigned by the task. In essence, the waiting time of a workflow is equal to the sum of the execution time of all preceding workflows in the queue. If there is no workflow in the queue at the time the workflow is submitted, the wait time is 0.

3.3. Workflow time feature extraction using SA

In a DRL-based method, the DRL agent needs to learn the characteristics of workflows and VMs in the cloud system to make a decision, and this requires an awareness of the state space of the environment. However, using a task relation matrix with the amount of computation of each task, the configurations and queue information of each VM as the state space would result in a huge feature space for the neural network of the DRL agent to learn, leading to long learning time and convergence problems. To address this, we propose to use the optimal execution time of a workflow on a cloud server as its feature, which can be used to define the required state space.

The SA algorithm is utilized to extract workflow features. Our scheme is also compatible with other metaheuristics methods, providing users with the flexibility to choose the most suitable approach for their specific requirements. To begin with, we employ the SA algorithm to establish the optimal execution order of subtasks within the workflow. Subsequently, based on this determined order, we derive the corresponding optimal execution time. This time, considered a significant feature of the workflow, plays a crucial role in subsequent task scheduling processes. Specifically, we denote the subtasks of a workflow as $\{SJ_2, SJ_0, SJ_1, ..., SJ_N\}$, where SJ_i is the ID of subtask i. The SA algorithm model for solving the optimal subtask execution order is as follows.

3.3.1. Solution space

Solution space S refers to the set of all possible execution orders that can successfully execute each task. A solution can be expressed as a route list, denoted as $routeList_i = \{SJ_2, SJ_0, SJ_1, .., SJ_N\}$, which indicates the order in which the subtasks are to be executed. Specifically, SJ_2 is executed first, followed by other subtasks, until SJ_N is completed. In this case, we define the execution order of subtasks as

$$routeList_i = \{SJ_2, SJ_0, SJ_1, .., SJ_N\} \tag{5}$$

3.3.2. Objective function

The objective function can be defined as the total time required to execute all subtasks in the current order, which is also known as the cost function.

$$totalTime_i = getTexe(routeList_i) \tag{6}$$

where *getTexe* computes the time required to complete all subtasks following a given execution order S. The optimal execution order is the one that results in the minimum value of the objective function.

3.3.3. New path generation

One way to generate a new subtask execution order is randomly selecting two subtasks of a workflow and swapping their positions to form a new order.

3.3.4. Cost function difference

$$\Delta f = totalTime_{n+1} - totalTime_n \tag{7}$$

where $totalTime_n$ is the time to complete all subtasks on the n-th path. Δf is the difference of the total execution time between two solutions, and then the acceptance probability P of the system from order n to $n+1$ is calculated by

$$P = \begin{cases} 1, & \Delta f \geq 0 \\ e^{-\frac{\Delta f}{T}}, & \Delta f < 0 \end{cases} \tag{8}$$

Here, T is the current temperature. When $\Delta f \geq 0$, the new solution is directly accepted; otherwise, the new solution is accepted with probability P to avoid falling into local optima.

4. The proposed SA-DQN

In this section, we present the proposed SA-DQN method and provide a detailed description of its implementation.

4.1. Background of DRL

DRL is a type of machine learning that utilizes interactive strategies to solve problems within an environment [33]. The fundamental components of the RL system consist of the agent, state, action, reward, and environment. The agent acquires information from the environment and perceives the current state s_t. Based on this information, the agent selects an action from a given action space. The agent's action changes

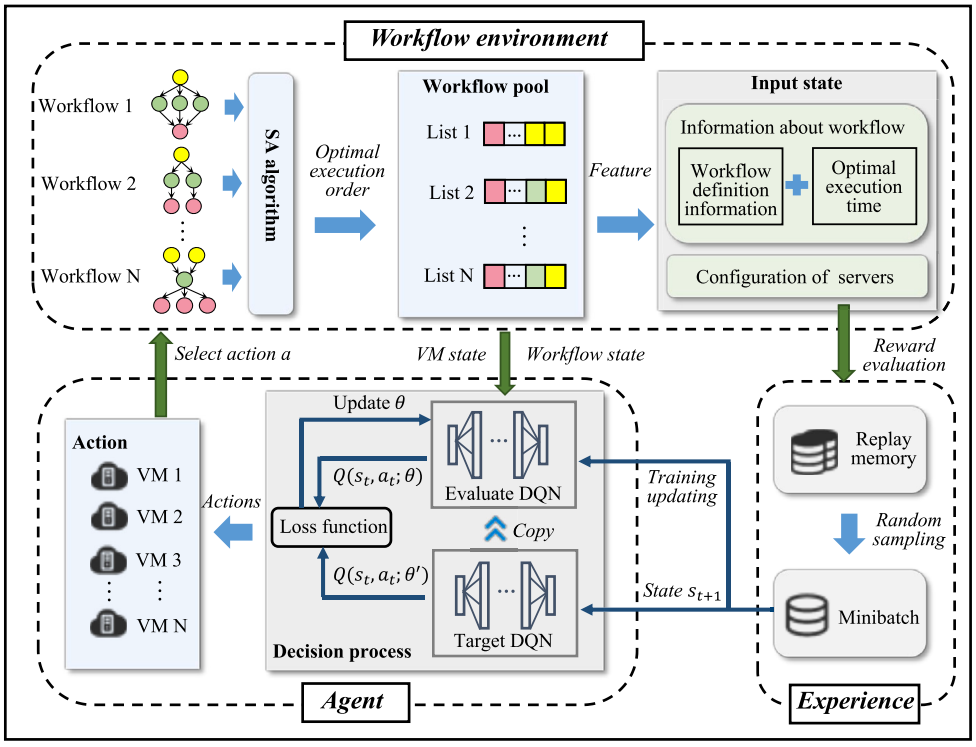


Fig. 3. The framework of SA-DQN.

the state of the environment via a state transfer function and receives a reward. Through continuous interactions with the environment and the receipt of rewards, the agent can learn and adapt decision-making strategies that maximize cumulative rewards.

Q-learning is a popular RL algorithm used to make rational decisions based on experiences. It involves an agent saving a value $Q(s,a)$ for each state s and its corresponding action a, representing the expected income obtained by taking action a at a certain time. The agent can use a value function to estimate the Q value of each action in the current state and make a decision action to maximize the long-term reward. After each action, the value function is updated iteratively using the following expression:

$$Q(s,a) \leftarrow Q'(s,a) + \alpha[r + \gamma * max_{a'} Q(s',a')] \tag{9}$$

where $\alpha \in [0, 1]$ is the learning rate, r is the reward after taking an action, $\gamma \in [0,1]$ is the discount factor, $Q(s,a)$ represents the new Q value, $Q'(s,a)$ represents the old Q value.

However, the traditional Q-learning algorithm has performance issues when dealing with complex problems with high-dimensional state and action spaces. To address this, the Deep Q-Network (DQN) uses a Deep Neural Network (DNN) to establish the correlation between each state-action pair and its associated value function. This allows for more efficient learning and decision-making. A diagram of the relevant DRL method is demonstrated in Fig. 3.

4.2. Introduction of SA

Our study utilizes the SA algorithm [34] to extract the time-dimension features of a workflow. The SA method is based on the principle of solid annealing, where a solid is heated to a high temperature and then gradually cooled. As the temperature increases, the particles become disordered, leading to an increase in internal energy. Conversely, as the particles cool, they become ordered and reach equilibriums at each temperature. At room temperature, the particles reach the ground state, minimizing the internal energy.

Algorithm 1 Simulated annealing.

```
Initialize temperature T (sufficiently large), initial solution state S (BestTime expected execution time in random subtask execution sequence), iteration times of each T value L.
Input: Job_i, VM_j
if T < T_min then
    Probability p of accepting the inferior solution is calculated based on the current temperature. The lower the temperature T is, the smaller p is.
    for k = 1, k < L; k++ do
        The execution sequence of the two subtasks is randomly swapped, and currentTime is calculated;
        if currentTime < BestTime then
            Accept new solutions;
        else
            Accept the new solution based on probability P;
        end if
    end for
    Perform cooling operation;
end if
Return the optimal solution;
```

The SA algorithm begins with a higher initial temperature and explores the global optimal solution of the objective function randomly by decreasing the temperature parameters while utilizing the probability jump characteristic. This approach enables the algorithm to escape local optimal solutions and converge to the global optimal solution. The detailed implementation of the algorithm is presented in Algorithm 1.

4.3. SA-DQN for workflow scheduling

In our workflow scheduling approach for cloud instances, we begin by utilizing the SA algorithm to determine the optimal execution time of the workflow on each cloud instance. Subsequently, we incorporate the execution time as a feature of the workflow in the state space of the scheduling system and feed them to the DRL model for training.

4.3.1. Action space

As described in the system scheduling model discussed in Section 3, a batch of cloud instances is rented, and a random number of workflows

arrive at each time point. The goal of our scheduling approach is to allocate each workflow to a designated instance. Therefore, the action space of the DRL model is defined as follows:

$$a = \left[VM_1, VM_2, ..., VM_N\right] \tag{10}$$

where VM_i represents the i-th cloud instance.

4.3.2. State space

When a user submits a workflow job, we employ the SA algorithm to obtain the optimal execution time of the job on each cloud instance. Specifically, for a given Job_j and a virtual machine VM_i, we obtain the execution time T_j^{exe} using the following equation.

$$T_j^{exe} = simulateAnnea(Job_j, VM_i) \tag{11}$$

In the equation above, *simulateAnnea* represents the SA function utilized to calculate the job execution time. Using this approach, when a workflow j arrives at time T_j, we can define the state of the system as follows, including both the states of the workflows and underlying VMs.

$$S_{tj} = \left[T_{j1}^{exe}, T_{j2}^{exe}, ..., T_{jn}^{exe}, T_{j1}^{wait}, T_{j2}^{wait}, ..., T_{jn}^{wait}\right] \tag{12}$$

where T_{jn}^{exe} represents the optimal execution time of workflow j on the n-th cloud instance, T_{jn}^{wait} is the waiting time of the n-th VM when job j arrives.

4.3.3. Reward function

Rewards play a crucial role in guiding agents to make effective decisions to achieve our scheduling goals. Specifically, our objective is to minimize the cost of job execution while satisfying the user's QoS_j, which is defined as the level of satisfaction of the user expressed as the proportion of the total execution time to total response time. Therefore, we can define QoS_j as follows:

$$QoS_j = \frac{T_j^{exe}}{T_j^{rep}} \tag{13}$$

where T_j^{rep} is the response time of job j, and T_j^{exe} is the total execution time of j. For j, we define the reward as

$$r = QoS_j + e^{-VM_{COST} * T_j^{exe}} \tag{14}$$

Here, VM_{COST} represents the cost per unit time of the cloud instance, and T_j^{exe} denotes the execution time of the job. Consequently, expression $VM_{COST} * T_j^{exe}$ signifies the execution cost of j. It is evident that the design of the reward function aligns with our optimization goals. Specifically, lower execution costs and higher QoS for the workflow will result in higher rewards. This incentivizes the agent to make decisions that minimize the cost of job execution while ensuring the timely completion of tasks, which is essential for satisfying QoS_j.

4.4. Model training

In a DQN algorithm, the DRL agent needs to select an appropriate action based on the estimated Q value derived from the DNN when making a decision given the current state. During the training process presented in Algorithm 2, we employ the $\xi - greedy$ strategy. Here, the selection probability of random instances is set to ξ, while the selection probability of instances with higher Q values is set to $1 - \xi$. This probability decreases as training time increases until it reaches a minimum value ξ_0. This approach encourages the DRL agent to initially explore randomly and gradually favor actions that yield higher returns as it learns.

In the DRL training process, the DNN learns from historical data to obtain a more accurate Q function using the experience replay mechanism. For each action a_t taken in state S_t, there is a corresponding reward r_t and the next state S_{t+1}. These values (S_t, a_t, r_t, S_{t+1}) are stored and periodically updated randomly to train the DQN model.

Algorithm 2 The training process of the proposed SA-DQN.

```
Initialize ε, α, γ learning frequency f, start learning time τ, minibatch SΔ, replay period η.
Initialize Q(s,a) to 0;
for each episode do
    Assign current state to s_t;
    for each step of episode do
        Choose an action (cloud instance) randomly with probability ε, or choose argmax_a Q(s,a;θ);
        Add task t to instance queue;
        Obtain reward r for action selected;
        Update Q(s,a) ← Q'(s,a) + α[r + γ * max_a' Q(s',a');
        Update s_t to s_{t+1};
        Update ε;
        Until state is the last state;
    end for
end for
```

Table 3
Summary of VM configurations.

VM_{id}	*CPU Num.*	*CPU Com.*	Cost
1	3	50	6
2	3	100	9
3	4	100	12
4	3	200	15
5	4	200	20
6	3	500	33

5. Performance evaluation

In this section, we compare our algorithm with several widely used real-time workflow scheduling methods, which have been conventionally employed in various scenarios.

5.1. Experimental framework

5.1.1. Simulation environment

The experiments are performed using the Python 3.8 environment on an Intel(R) Core(TM) i5-8265U CPU platform. Referring to previous work [35], we set up cloud instances that resemble the instances defined by real-world IaaS providers. Specifically, we assume that we have 6 available VMs. These VMs are distinguished by their on-demand usage cost, the number of CPUs or cores (i.e., *CPU Num*), and their computational performance (i.e., *CPU Com*). The configuration of the 6 instances used in our experiments, unless otherwise stated, is presented in Table 3. Moreover, to simulate a real-world scenario, we assign higher costs to VMs with greater performance. Accordingly, we utilize 3 low-cost instances, 2 medium-cost instances, and 1 high-cost instance. We evaluate our approach using multiple types of workflows, and Fig. 4 depicts the three types used in our evaluations.

In our evaluation, we initialize the underlying DQN using a feedforward neural network consisting of a hidden layer with 50 neurons. The replay memory is set to 500 and mini-batch is set to 30. The learning rate is set to 0.01 and the target iteration is set to 50 decisions per set. Moreover, other parameters are set to: $\gamma = 0.9$, $F = 1$, $\tau = 500$, and in each learning iteration, ξ decreases by 0.002 from 0.9.

5.1.2. Baseline solutions

To assess the performance of our SA-DQN, we compare it with 4 baseline methods: random scheduling, round-robin scheduling, earliest scheduling, and SA-GA scheduling. The random scheduling approach (referred to as Random) is a straightforward strategy that randomly selects a VM instance for each job. The round-robin scheduling approach (referred to as RR) aims to evenly distribute jobs across VM instances by selecting them in a circular order. The earliest scheduling method (referred to as Earliest) is a time-greedy approach in which incoming jobs are allocated to the earliest idle VM instance. Hence, the earliest scheduling method assigns new jobs to the VM instance with the short-

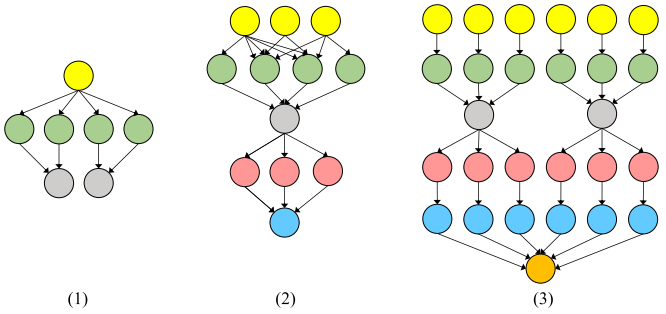


Fig. 4. Three types of workflow tasks used in our evaluation.

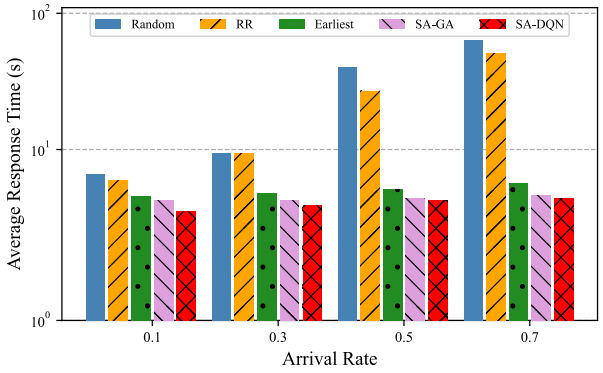
est wait time [36]. SA-GA refers to an approach where the SA method is employed to obtain the optimal execution time for workflows, and the GA is used to schedule processed workflows. Within the context of the GA, the optimization objective is to improve QoS.

5.2. Comparison of scheduling algorithms

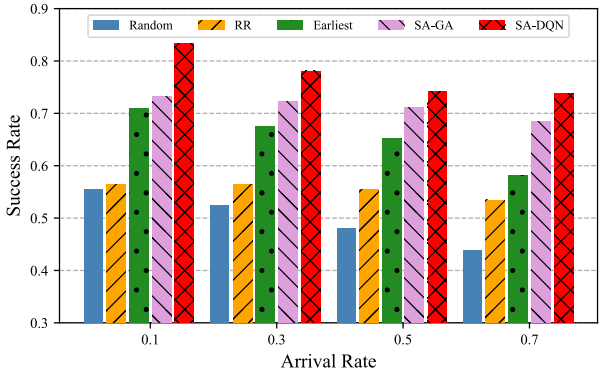
5.2.1. Varying mean arrival rates

In this experiment, we evaluate the performance of the 5 methods under varying levels of job arrival intensities (hereinafter referred to as arrival rates). To this end, we classify the arrival rates into 4 scenarios: 0.1, 0.3, 0.5, and 0.7. A higher arrival rate signifies a greater frequency of job arrivals. Additionally, we set the job size range to [200, 300], the number of sub-tasks to [10, 20], and the depth to [2, 3].

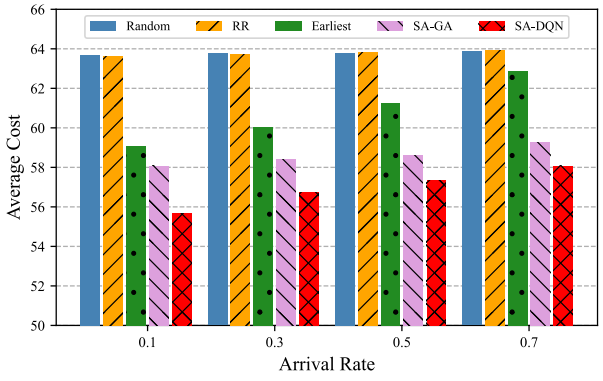
In this paper, it is worth noting that the purpose of the SA algorithm is to obtain features of the workflow. Therefore, the average response time and cost of workflow scheduling reported in this study encompass solely the process of scheduling the workflow onto the most appropriate VM, excluding the invocation of the SA algorithm. The experimental results are presented in Fig. 5, which show that our SA-DQN approach significantly outperforms the other baseline methods. Fig. 5(a) demonstrates that SA-DQN consistently achieves the shortest response time among almost all methods. In Fig. 5(b), with the increase in the arrival rate, the success rates of all algorithms exhibit a general decline. Nevertheless, the SA-DQN algorithm consistently outperforms the other algorithms, maintaining the highest success rate across all arrival rate levels. Specifically, it approximately a approximately 50% improvement compared to Random, around a 40% improvement compared to RR, roughly a 20% improvement compared to Earliest, and about a 10% improvement compared to SA-GA. Moreover, Fig. 5(c) indicates that our approach can achieve the lowest average cost, and there is an overall upward trend in the average cost of all algorithms as the arrival rate increases.



(a) Average Response Time



(b) Success Rate



(c) Average Cost

Fig. 5. Performance comparison by varying arrival rates.

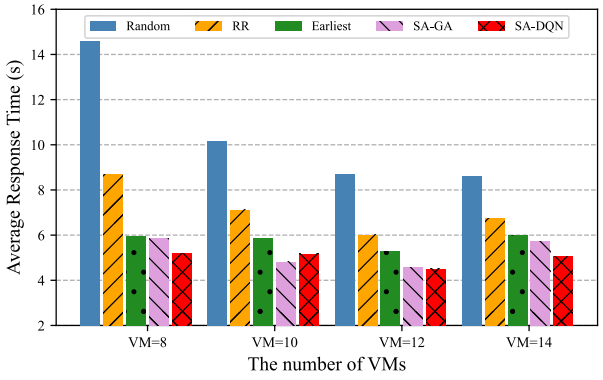
5.2.2. Varying numbers of VMs

In this experiment, we assess the performance of the 5 scheduling algorithms with different numbers of VMs. The experimental outcomes are illustrated in Fig. 6. We consider 4 distinct settings for the number of virtual machines: 8, 10, 12, and 14. The workloads and types of workflows are consistent with the settings in Section 5.2.1. Additionally, to ensure a fair comparison, we fix the job arrival rate at 0.5 in this experiment.

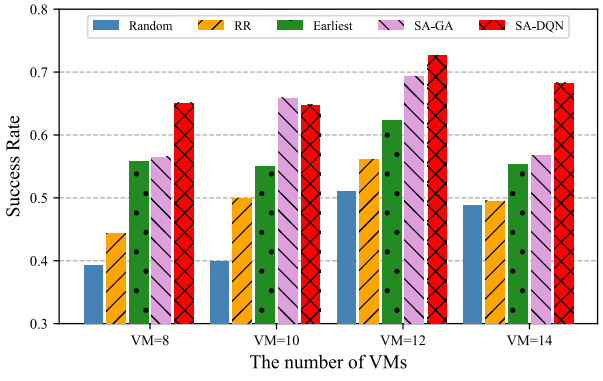
Based on the results shown in Fig. 6, our proposed SA-DQN approach consistently achieves optimal performance in terms of the average response time, except when the number of VMs is 10. Furthermore, it significantly outperforms other comparison algorithms in terms of the success rate and average cost. Additionally, as the number of VMs increases, all algorithms exhibit an initial decline followed by a tendency to plateau or slightly increase in the average response time and average cost. In contrast, as for the success rate, they show an initial increase followed by a region of plateau or even a slight decrease. From these observations, we can conclude that for workflow scheduling, appropriately increasing the number of VMs can enhance scheduling performance. However, it is essential to note that blindly increasing the number of VMs does not guarantee the optimal performance, and the specific quantity should be determined based on the scenario and requirements.

5.2.3. Varying workload complexities

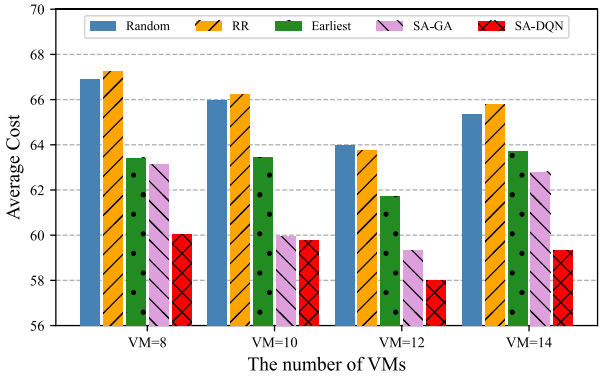
In this experiment, we evaluate the performance of the 5 scheduling methods under different workloads. We generate 3 types of workloads, each with varying levels of complexities, as shown in Table 4. In our study, we adopt a specific approach to generate the workflow structure. First, we randomly select the number of depths and the number of subtasks within each depth from the predefined range. Subsequently, we establish the dependencies between the subtasks through a random process. Once the workflow structure is determined, we proceed to randomly assign workloads to each subtask within the defined range. These



(a) Average Response Time

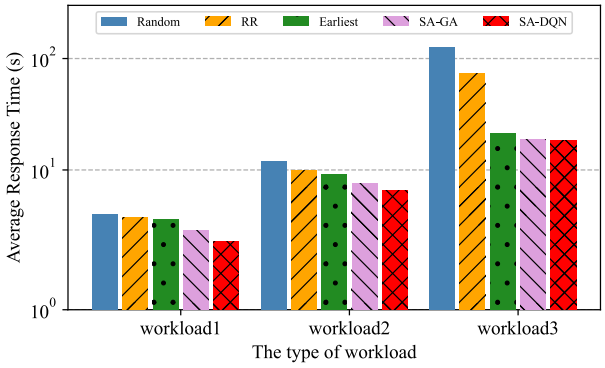


(b) Success Rate

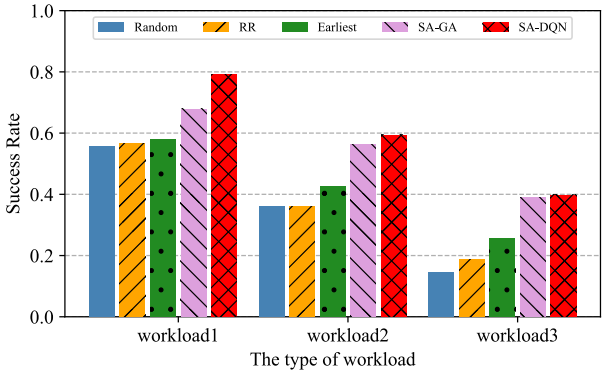


(c) Average Cost

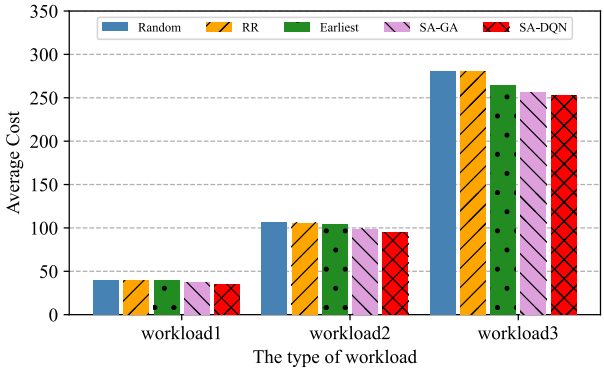
Fig. 6. Performance comparison by varying numbers of VMs.



(a) Average Response Time



(b) Success Rate



(c) Average Cost

Fig. 7. Performance comparison among varying workloads.

Table 4
Summary of different workloads.

	Depths	Number of subtasks	Workload of subtasks
Workload 1	[2, 3]	[5, 10]	[100, 200]
Workload 2	[4, 5]	[10, 20]	[200, 300]
Workload 3	[6, 7]	[20, 30]	[300, 400]

workloads represent different workflows with different levels of complexities. Without loss of generality, we set the arrival rate to 0.1, and the results are reported in Fig. 7.

Based on the results depicted in Fig. 7, it is evident that as the complexity of the workload increases, workflow scheduling methods generally require higher average response time, lower success rates, and higher average costs. However, regardless of the workload type, SA-DQN consistently achieves superior performance in terms of the average response time, success rate, and average cost. Particularly, as for the success rate, SA-DQN significantly outperforms other comparison algorithms. Based on this, we can conclude that SA-DQN is capable of adapting to workflows with varying complexities, ensuring consistent and excellent performance.

5.2.4. Experiments on real-world workflows

To assess the practical applicability of our proposed method in workflow scheduling, we conduct a comprehensive evaluation that incorporates real-world workflows. In this evaluation, we meticulously generate a total of 500 workflows, with each workflow's structure randomly selected from the 3 structures illustrated in Fig. 8. These authentic workflows are sourced from the well-established Pegasus project. Our evaluation aims to gauge the effectiveness of the proposed algorithm in comparison to other existing algorithms, under varying arrival rate conditions. Specifically, we consider the arrival rates of 0.1, 0.3,

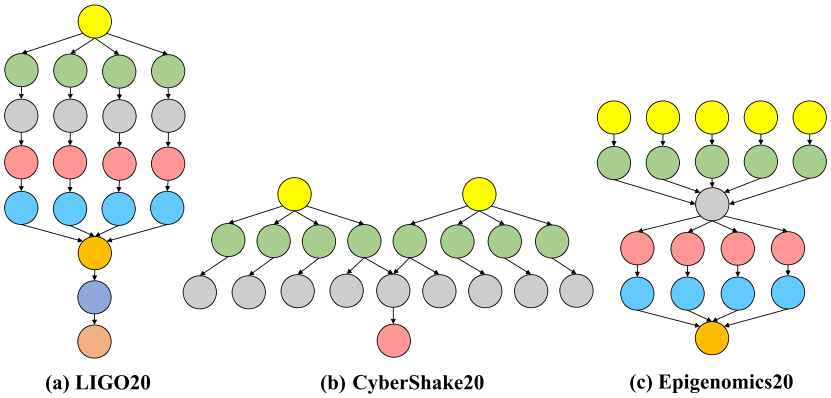


Fig. 8. Three types of real-world workflow tasks used in our evaluation.

0.5, and 0.7, enabling us to comprehensively analyze the algorithm's performance. The outcomes of our experiments are depicted in Fig. 9.

The results from Fig. 9 demonstrate that SA-DQN consistently outperforms other comparison algorithms in terms of the average response time, success rate, and average cost across all arrival rate levels. Particularly noteworthy is its substantial advantage in the success rate, achieving approximately a 60% improvement compared to Random, about a 50% improvement compared to RR, roughly a 20% improvement compared to Earliest, and approximately a 10% improvement compared to SA-GA. Furthermore, as the arrival rate increases, all scheduling methods show an upward trend in the average response time and average cost, while the success rates exhibit a declining trend. These observations lead to the conclusion that SA-DQN exhibits practical applicability and reliability.

6. Conclusion

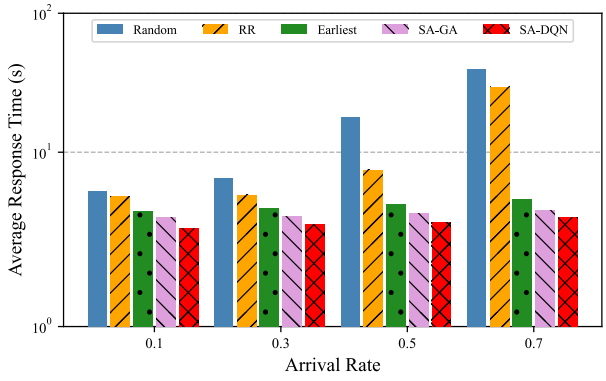
In this work, we propose SA-DQN, an effective approach for scheduling real-time workflows in cloud computing. Our approach seamlessly integrates metaheuristic and DRL algorithms to optimize job execution costs while ensuring job response time and success rates. To facilitate the DRL process, we employ the SA algorithm to extract task and cloud server features. We provide a detailed implementation of our method, and our experimental results demonstrate that SA-DQN outperforms existing approaches in the presence of different workloads. In future work, we plan to employ a hierarchical structure with multiple DRL agents to handle workflow scheduling problems in large-scale scenarios, such as those with hundreds of cloud instances. This would enable us to further improve the performance of SA-DQN and extend its applicability to more complex cloud environments.

CRediT authorship contribution statement

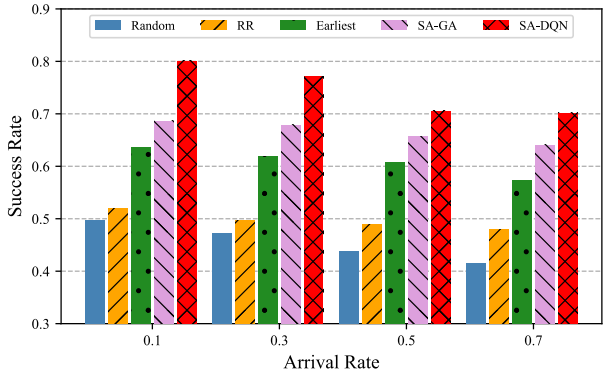
Yan Gu: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Feng Cheng:** Conceptualization, Data curation, Supervision, Validation, Writing – review & editing. **Lijie Yang:** Data curation, Formal analysis, Resources, Writing – review & editing. **Junhui Xu:** Data curation, Formal analysis, Validation, Writing – review & editing. **Xiaomin Chen:** Investigation, Validation, Visualization, Writing – review & editing. **Long Cheng:** Conceptualization, Data curation, Formal analysis, Investigation, Validation, Visualization, Writing – review & editing.

Declaration of competing interest

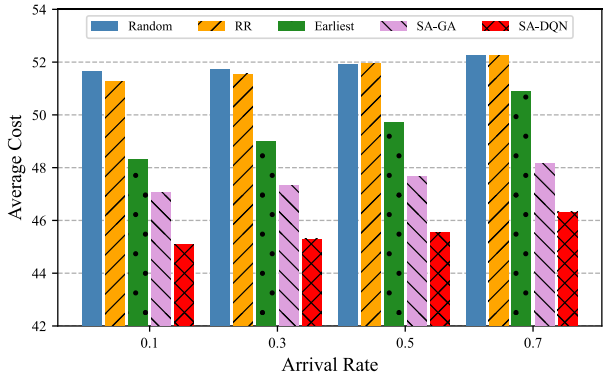
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.



(a) Average Response Time



(b) Success Rate



(c) Average Cost

Fig. 9. Performance comparison on real-world workflows.

Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities (2023JC004 and 2023YQ002).

References

[1] L. Cheng, B.F. van Dongen, W.M. van der Aalst, Scalable discovery of hybrid process models in a cloud computing environment, IEEE Trans. Serv. Comput. 13 (2) (2019) 368–380.
[2] J. Liu, H. Shen, H. Chi, H.S. Narman, Y. Yang, L. Cheng, W. Chung, A low-cost multi-failure resilient replication scheme for high-data availability in cloud storage, IEEE/ACM Trans. Netw. 29 (4) (2020) 1436–1451.
[3] 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.
[4] Y. Mao, W. Yan, Y. Song, Y. Zeng, M. Chen, L. Cheng, Q. Liu, Differentiate quality of experience scheduling for deep learning inferences with docker containers in the cloud, IEEE Trans. Cloud Comput. 11 (2) (2022) 1667–1677.
[5] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, E.-g. Talbi, Towards understanding uncertainty in cloud computing resource provisioning, Proc. Comput. Sci. 51 (2015) 1772–1781.
[6] Y. Yu, V. Jindal, I.-L. Yen, F. Bastani, Integrating clustering and learning for improved workload prediction in the cloud, in: Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing, IEEE, 2016, pp. 876–879.
[7] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, B. Hertzberger, Distributed execution of aggregated multi domain workflows using an agent framework, in: Proceedings of the 2007 IEEE Congress on Services, IEEE, 2007, pp. 183–190.
[8] X. Ma, H. Xu, H. Gao, M. Bian, Real-time multiple-workflow scheduling in cloud environments, IEEE Trans. Netw. Serv. Manag. 18 (4) (2021) 4002–4018.
[9] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.
[10] Y. Hu, C. de Laat, Z. Zhao, Learning workflow scheduling on multi-resource clusters, in: Proceedings of the 2019 IEEE International Conference on Networking, Architecture and Storage, IEEE, 2019, pp. 1–8.
[11] S.Z. Selim, K. Alsultan, A simulated annealing algorithm for the clustering problem, Pattern Recognit. 24 (10) (1991) 1003–1008.
[12] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, in: Proceedings of the AAAI Conference on Artificial Intelligence, ACM, 2018, pp. 3207–3214.
[13] M.H. Shirvani, A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems, Eng. Appl. Artif. Intell. 90 (2020) 103501.
[14] Y.-H. Jia, W.-N. Chen, H. Yuan, T. Gu, H. Zhang, Y. Gao, J. Zhang, An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization, IEEE Trans. Syst. Man Cybern. Syst. 51 (1) (2018) 634–649.
[15] H.Y. Shishido, J.C. Estrella, C.F.M. Toledo, M.S. Arantes, Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds, Comput. Electr. Eng. 69 (2018) 378–394.
[16] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, S. Hu, Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft, Future Gener. Comput. Syst. 93 (2019) 278–289.
[17] A. Jayanetti, S. Halgamuge, R. Buyya, Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments, Future Gener. Comput. Syst. 137 (2022) 14–30.
[18] R. Xie, D. Gu, Q. Tang, T. Huang, F.R. Yu, Workflow scheduling in serverless edge computing for the industrial Internet of things: a learning approach, IEEE Trans. Ind. Inform. 19 (7) (2022) 8242–8252.
[19] Y. Xiang, X. Yang, Y. Sun, H. Luo, A fault-tolerant and cost-efficient workflow scheduling approach based on deep reinforcement learning for it operation and maintenance, in: Proceedings of the 2023 26th International Conference on Computer Supported Cooperative Work in Design, IEEE, 2023, pp. 411–416.
[20] P. Guo, Z. Xue, An adaptive pso-based real-time workflow scheduling algorithm in cloud systems, in: Proceedings of the 2017 IEEE 17th International Conference on Communication Technology, IEEE, 2017, pp. 1932–1936.
[21] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, N. Najjari, Online multi-workflow scheduling under uncertain task execution time in iaas clouds, IEEE Trans. Cloud Comput. 9 (3) (2019) 1180–1194.
[22] H. Chen, X. Zhu, D. Qiu, L. Liu, Uncertainty-aware real-time workflow scheduling in the cloud, in: Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing, IEEE, 2016, pp. 577–584.
[23] F. Song, Y. Ma, I. You, H. Zhang, Smart collaborative evolvement for virtual group creation in customized industrial IoT, IEEE Trans. Netw. Sci. Eng. 10 (5) (2022) 2514–2524.
[24] F. Song, M. Zhu, Y. Zhou, I. You, H. Zhang, Smart collaborative tracking for ubiquitous power IoT in edge-cloud interplay domain, IEEE Int. Things J. 7 (7) (2019) 6046–6055.
[25] P. Zhang, M. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 772–783.
[26] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, J. Murphy, A woa-based optimization approach for task scheduling in cloud computing systems, IEEE Syst. J. 14 (3) (2020) 3117–3128.
[27] M. Adhikari, T. Amgoth, S.N. Srirama, A survey on scheduling strategies for workflows in cloud environment and emerging trends, ACM Comput. Surv. 52 (4) (2019) 1–36.
[28] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, H. Huang, ET2FA: a hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud, IEEE Trans. Serv. Comput. 16 (3) (2022) 1807–1821.
[29] J. Guo, L. Cheng, S. Wang, CoTV: cooperative control for traffic light signals and connected autonomous vehicles using deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. 24 (10) (2023) 10501–10512.
[30] Q. Liu, L. Cheng, A.L. Jia, C. Liu, Deep reinforcement learning for communication flow control in wireless mesh networks, IEEE Netw. 35 (2) (2021) 112–119.
[31] M. Mekala, G. Dhiman, G. Srivastava, Z. Nain, H. Zhang, W. Viriyasitavat, G. Varma, A DRL-based service offloading approach using dag for edge computational orchestration, IEEE Trans. Comput. Soc. Syst. (3) (2024) 3070–3078.
[32] L. Wang, E. Gelenbe, Adaptive dispatching of tasks in the cloud, IEEE Trans. Cloud Comput. 6 (1) (2015) 33–45.
[33] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: a brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38.
[34] E.A. Avello, F.F. Baesler, R.J. Moraga, A meta-heuristic based on simulated annealing for solving multiple-objective problems in simulation optimization, in: Proceedings of the 2004 Winter Simulation Conference, IEEE, 2004.
[35] M. Ghobaei-Arani, S. Jabbehdari, M.A. Pourmina, An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach, Future Gener. Comput. Syst. 78 (2018) 191–210.
[36] Y. Wei, L. Pan, S. Liu, L. Wu, X. Meng, DRL-scheduling: an intelligent qos-aware job scheduling framework for applications in clouds, IEEE Access 6 (2018) 55112–55125.