Scientia Iranica (2024) 31(5), 373–387

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
https://scientiairanica.sharif.edu

A self-adaptive approach to job scheduling in cloud computing environments

A. Sheibanirad and M. Ashtiani*

Cloud Computing Center, School of Computer Engineering, Iran University of Science and Technology, Tehran, P.O. Box 1684613114, Iran.

Received 30 September 2021; received in revised form 7 April 2023; accepted 21 November 2023

KEYWORDS
Cloud computing;
Reinforcement learning;
Job scheduling;
Autonomicity;
Soft actor-critic.

Abstract. Manual configuration of available resources in the data center, as well as manual decision-making for customers' requests, makes the resource management process potentially error-prone. Therefore, the resource manager should make intelligent decisions for assigning available resources to existing requests to ensure scalable and efficient on-demand resource provisioning. Cloud job scheduling mechanisms aim to allocate the resources to users' submitted jobs optimally, yet optimal scheduling is an NP-complete problem. To address these challenges, many researchers have tried to tackle the job scheduling problem by proposing automatic solutions using Reinforcement Learning (RL) methods. Unfortunately, most of these methods ignore fair response time to all the incoming jobs with the proper utilization of data center resources. In this research, we use deep RL as a sequential decision-making method for automatic resource management that changes its behavior to deal with environmental changes. The approach uses the discrete soft-actor-critic algorithm. It has efficient sampling and stable learning convergence, as well as a precise adjustment of learning hyperparameters. Results show that compared to DeepRM and DeepScheduler, our approach improves slowdown and the balance of slowdown by at least three times using Google's dataset.

1. Introduction

A massive number of jobs are submitted daily to cloud systems which need to be mapped to the existing resources simultaneously and efficiently. Therefore, devising an appropriate approach that minimizes the jobs' execution delay and balances the cloud resource utilization is crucial. The advent of cloud automation reduces manual intervention and improves resource management in large-scale workloads. It encompasses designing automation techniques and tools for resource allocation and management that execute on top of the virtualized cloud environment to make real-time decisions [1].

One of the challenges in cloud computing is job scheduling, which aims to minimize the response time of required jobs based on user-defined job profiles. Scheduling in cloud computing has two points of view: (1) the users' viewpoint and (2) the providers'

*. *Corresponding author. Tel.: +98 21 73225328;*
E-mail addresses: sheibanirad@mail.mui.ac.ir (A. Sheibanirad); m_ashtiani@iust.ac.ir (M. Ashtiani)

To cite this article:
A. Sheibanirad, and M. Ashtiani, "A self-adaptive approach to job scheduling in cloud computing environments", *Scientia Iranica* (2024), **31**(5), pp. 373-387
DOI: 10.24200/sci.2023.59168.6090

viewpoint. Job scheduling cannot be solved using linear programming approaches, which means it is impossible to find an ideal solution (it is an NP-complete problem). Several branches and bound approximation methods have been proposed to deal with this issue [2].

Autonomic cloud computing approaches are potential solutions for effective resource allocation by realizing users' Quality Of Service (QoS) requirements and handling unexpected failures at runtime to optimize QoS parameters [3].

Efficient job scheduling faces heterogeneous complexity. The policy of the scheduler significantly depends on understanding the workload and environment. Hence, the scheduler needs to consider the execution time of each distinct workload and the overall performance based on the type of workload, namely heterogeneous workloads (i.e., different QoS requirements) and homogenous workloads (i.e., similar QoS requirements) [4].

Currently, most research approaches use heuristics that are applied to solve job scheduling for different optimization objectives. These heuristics depend on domain parameters defined by experts and optimizing performance metrics in a principled manner is notoriously challenging [5]. The recent development of artificial intelligence evinces new automation methods to help reach an optimal situation in a dynamic and complex environment like the cloud. In this work, we propose an approach that schedules batch, unpredictable, and diverse incoming jobs online and dynamically. We have designed the proposed scheduler based on Reinforcement Learning (RL). In this research, self-adaptation and system efficiency are realized by performing continuous monitoring, understanding the characteristics of the environment and its potential changes as well as dynamically and automatically changing the scheduler's policy. The scheduler uses the soft-actor-critic algorithm [6] which is sample efficient, considers automatic management, and uses curiosity learning to explore a variety of states. The proposed load-sensitive scheduler focuses on response time and the fair assignment of resources to the existing jobs. The rest of the paper is structured as follows: Section 2 gives a comprehensive overview of related work, discussing the pros and cons of existing approaches. Section 3 presents the details of the proposed approach, consisting of the architecture, main components, and algorithms. In Section 4, the evaluation results are provided. Some of the practical limitations and real-world applicability challenges are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related work

Traditionally, researchers have used heuristics such as fair scheduling [7,8], first-fit [9], simple packing strategies [10], and meta-heuristic algorithms like genetic algorithms [11] or ant colony optimization [12] to address resource management issues, which led the system to use its resources effectively by ensuring fairness and efficient scheduling. These simple heuristics prioritize generality to attain the best performance on a specific scenario and workload and regulate parameters by complex, tedious, and iterative tests [5,13].

For four reasons, RL is an acceptable approach for decision-making problems like job scheduling. Firstly, it can model and solve job scheduling within complex data center environments and decision-making policies as deep neural networks. Secondly, it works with unlabeled data. Thirdly, it can be trained using related policies and reward signals. Finally, its continuous and rapid learning capabilities enable agents to converge towards optimal conditions based on specified workloads [5].

For the first time, Mao et al. [5] proposed a simple multi-resource cluster scheduler (DeepRM), which manages incoming jobs based on the scheduler's policy and allocates them to the data center. DeepRM operates online, packing jobs with multiple resource demands. It tends to optimize the average slowdown or completion time as an objective function.

Unfortunately, this research suffers from a high variance problem, which leads to low accuracy when computing the gradient. Additionally, it relies on the Monte Carlo method to update the parameters, which requires massive calculations. Finally, DeepRM doesn't support a multi-resource multi-machine environment.

Mao et al. [14] solves the dependent job scheduling problem in a Spark cluster through a graph embedding technique. This approach is a general-purpose scheduling service for data processing jobs that can be used for interdependent tasks. The approach uses the Monte-Carlo method, which has low accuracy and requires massive calculations, as also mentioned for DeepRM.

Chen et al. [15] improved and extended the DeepRM approach to multiple server clusters through deep RL. This model reconstructs the state space representation, rewriting the reward function for the RL agent, using a convolutional input layer. On the other hand, this work suffers from a high variance problem. Also, updating the parameters requires massive calculations.

DeepRM2 [16] proposes a preemptive and multi-cluster model with two new ideas. The first idea is imitation learning, which helps the scheduler learn primitive policies before starting the training process using deep RL The second idea is to change the policy network's structure from a fully connected network to a convolutional neural network that facilitates image feature extraction. This approach can be used for online and offline resource scheduling.

CuSH [17] is a resource management and job

scheduling framework that uses deep neural networks and RL. This framework uses Neural Networks (NN) as automatic agents which learn to make optimal scheduling choices based on a dataset that contains the jobs' history and performance characteristics. The implementation consists of job and policy selector modules, each of which trains its NN with an RL configuration. The first module is the policy selector, and the second module is the job selector. Unfortunately, like previous research, this study suffers from a high variance problem and relies on the Monte Carlo method to update the parameters. Another downside is its focus on a single objective function.

SCARL [18] is an RL-based job scheduler developed to accommodate the diversity and heterogeneity of jobs and machines in a cluster. The model includes: (1) a cluster, (2) a scheduler based on RL, and (3) an attentive interpreter. SCARL considers two performance objectives: (1) maximizing job completion delay (i.e., slow down) and (2) supporting jobs with deadline constraints. However, this work suffers from several limitations. It faces a high variance problem, updating the parameters requires massive calculations, and its evaluation is only based on synthesized simulation scenarios. Attentive learning takes time and has a high overhead for the scheduler. Also, the RL-based scheduler cannot operate on large-scale workloads over multiple clusters.

DeepJS [19] is a job scheduling algorithm based on deep RL and designed to accommodate the bin packing problem. It calculates the fitness of each action to solve the bin packing problem, which describes how a machine and a task are mapped and minimizes the makespan. This research, like previous research, suffers from a high variance problem and relies on the Monte Carlo method to update the parameters which requires massive calculation.

Liang et al. [20] proposed a policy-value-based deep RL scheduling method called A2cScheduler [20]. A2cScheduler is based on an actor-critic method. The multi-step Temporal-Difference (TD) method is a key feature of the A2cScheduler for updating parameters, which improves the speed of the scheduler's training process compared to the conventional Monte Carlo methods. The objective function of this approach is to reduce the average job waiting time without any background knowledge. Thus, it learns the scheduling policies directly and automatically from past trajectories. This approach is the most relevant research to our work. Unlike this approach, our proposed scheduler can optimize more than one objective. Also, curiosity (i.e., entropy) as intrinsic motivation. The proposed structure of the algorithm leads the agent to discover rare events, have more efficient sampling, reduce the chance of halting in the local optimum, and learn dynamically.

Guo et al. [21] presented a deep RL-based solution (DeepRM-Plus) for the cloud resource management problem [21]. The authors used a supervised learning-based exploration called imitation learning. This research considers two objectives: (1) minimizing turnaround and (2) minimizing cycling time. In our approach, the critic adjusts the initial parameters without human intervention. Additionally, the Soft Actor-Critic Discrete (SACD) algorithm uses entropy to have more efficient sampling for examining the action space (using a non-random search and reduces the chance of halting in the local optimum).

Xu et al. [22] proposed a multi-factored scaling approach whose objectives are finding the best scaling technique based on execution cost and response time. This approach, termed CoScal, is a resource provisioning strategy in cloud environments that responds to potential workload fluctuations by scaling containerized microservices. Unlike other scheduling approaches, CoScal doesn't decide the initial placement of a containerized task [23]. Table 1 contains a summary of the related works discussed in this section.

3. The proposed approach

In this section, we will describe the architecture of the approach based on the MAPE-k and SACD algorithms [24]. The formulations and theoretical details, the model and formulation of the automatic scheduler based on the SACD, and the scheduling process and its algorithm based on the SACD are also given.

3.1. The components

In this part, we introduce the proposed cooperative and centralized resource manager. Multiple cloud customers send their requests to it to execute their jobs. The resource manager is responsible for collecting available resources in the data center and analyzing requests from customers. More importantly, it should make intelligent decisions regarding the assignment of the available resources to requests. To ensure scalable and efficient on-demand resource provisioning, the resource manager is designed to deliver a feasible and optimal resource scheduling scheme for customers.

The resource manager consists of two core components which are described as follows:

1. *Environment*: This component manages the monitoring and execution section of MAPE-K. The environmental information consists of the data center resource's state, the independent jobs in the queue, and the backlog. It is responsible for generating a job sequence, getting a new job from the sequence, and calculating the reward function for the jobs in the requested pool. It also observes the resources and the request pool and allocates the selected jobs to available resources. We have formulated the

Table 1. A comparative summary of the reinforcement learning-based resource management approaches.

Research article	Objectives	Multi-Cluster	Cluster config.	Scheduling type	Uncertainty	Learning method	Curiosity
DeepRM [5]	Minimizing slowdown or completion time	×	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm (Monte Carlo)	×
DeepRM+ [15]	Minimizing slowdown	√	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm. (Monte Carlo)	×
DeepRM2 [16]	Minimizing slowdown or completion time	√	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm. (Monte Carlo)	×
SCARL [18]	Minimizing slowdown	×	Heterogenous	non-preemptive	√	Attentive RL	×
CuSH [17]	Minimizing normalized turnaround time	√	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm. (Monte Carlo)	×
DeepJS [19]	Minimizing makespan and maximizing throughput	×	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm. (Monte Carlo)	×
DeepScheduler [20]	Minimizing slowdown and average waiting time and completion time	√	Unified pool	non-preemptive	√	DeepRL-ActorCritic algorithm. (Advantage-Actor -Critic)	×
DeepRM_Plus [21]	Minimizing the turnaround and cycling times	√	Unified pool	non-preemptive	√	DeepRL-Reinforce algorithm. (Monte Carlo) with imitation learning	×
Proposed approach	**Minimizing** slowdown and improving resource utilization	√	**Unified pool**	**non-preemptive**	√	**DeepRL-** Soft-ActorCritic algorithm.	√

changed status (the scheduler's input) as a distinct and comprehensive image. The status consists of the data center's resources and the queued jobs at time t (i.e., the incoming request queue and backlog). The details will be explained further in Section 4.3;

2. *Agent (SACD scheduler)*: This component manages the analysis and planning section of the MAPE-K, interacting with the environment. The scheduler receives the state of the environment and, based on its model (i.e., the SACD algorithm), selects proper actions. This component plays a key role in the resource manager's ability to schedule resources most appropriately.

3.2. The scheduler's architecture

The scheduler uses a SACD which consists of two interactive components: (1) the actor and (2) the critic. The actor adjusts the scheduler policy based on the

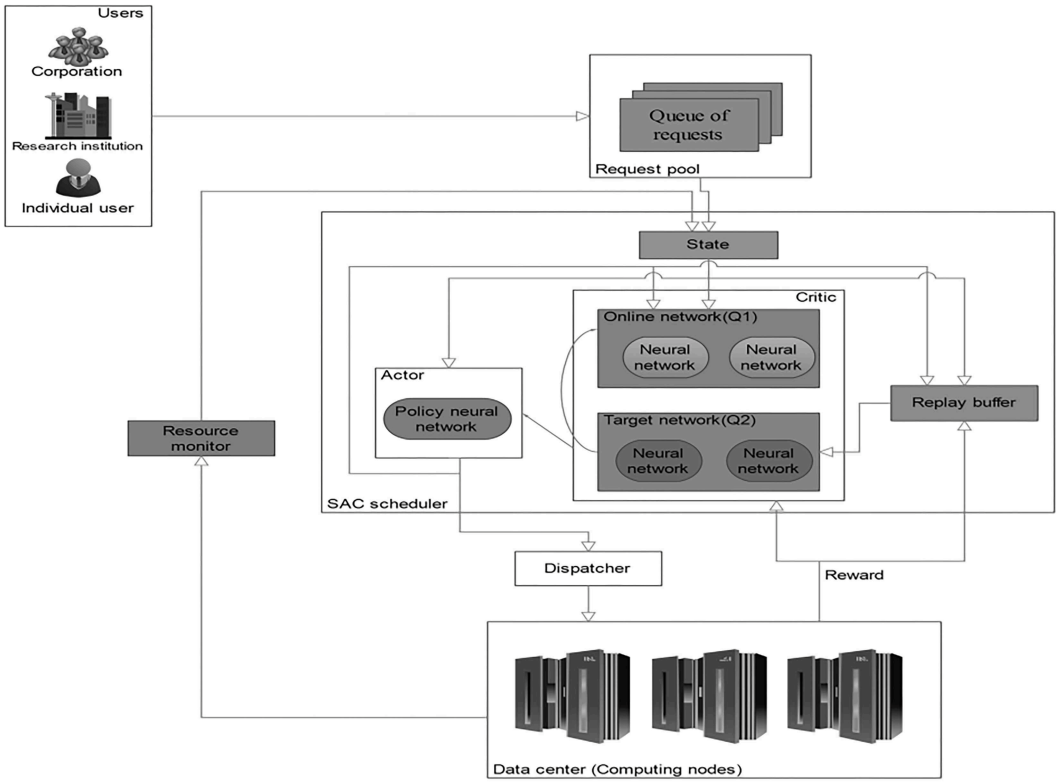

Figure 1. The proposed architecture for the scheduler, its components, and its interaction with the data center.

selected jobs from the queue, the available resources of the data center, and the reward function's goals. The critic automatically evaluates the actor's policies based on the state-action function. Based on this algorithm, the proposed scheduler interacts with the incoming jobs and converges to a stable situation in its training phase. Figure 1 shows the internal details of the proposed scheduler.

In Figure 1, the critic contains four NNs that correct or confirm the actor's behavior using the reward function and the previous trajectories in the replay buffer. The critic receives the state of the environment (i.e., data center state, the queue of incoming jobs, and the current backlog) to estimate the Q-value function and evaluate the actor's policy. The actor modifies its policy distribution based on the gradient descent of the critic state-value function and the entropy. This regulation makes the scheduler's policy more adaptive to its environment. Due to the nature of the SACD, the scheduler encounters a variety of incoming jobs and their fluctuations. Thus, it uses trial-and-error search to reach an optimal policy.

The proposed scheduler's main parts are: (1) the trained policy, (2) the objective (reward) function, (3) the replay buffer, and (4) the critic. The trained policy specifies the scheduler decision to select the incoming jobs. The objective (reward) function determines the quality and usefulness of the scheduler's chosen actions based on entropy and penalties related to delay, waiting, and increased job response time. The replay buffer stores the trajectories and provides them randomly to the scheduler. The critic dynamically evaluates the scheduler behavior according to the data center status and the reward function. The scheduler performs non-preemptive online job scheduling. This component receives the submitted jobs in its queue. Then, it maps the available resources based on the requested resource profile of each job and its objective function. It evaluates its learned policy by TD (0) error. Figure 2 shows the automated iterative scheduling process.

3.3. The proposed scheduling algorithm

The SAC algorithm is a model-free deep RL algorithm that has been successfully applied to a wide range of sequential decision-making and control tasks. It computes an optimal policy that maximizes both the long-term expected reward and the entropy of the policy. Policy entropy is a measure of policy uncertainty given by the state. A higher entropy value promotes more exploration. Maximizing both the reward and the entropy balances the exploration and exploitation of the environment [25].

During the training phase, a SAC agent:

1. Updates the actor and critic's properties at regular intervals;
2. Estimates the mean and standard deviation for selecting an action in the continuous action space and

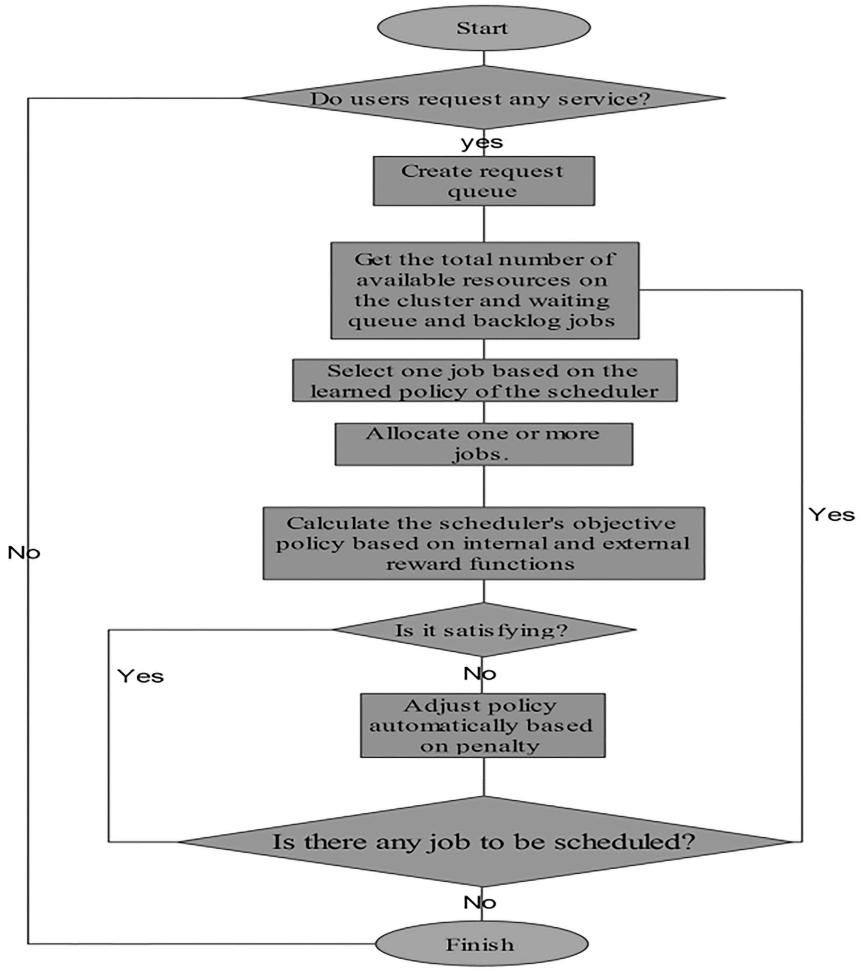

Figure 2. The scheduler's process for the incoming jobs.

randomly selects actions based on the probability distribution;

3. Updates an entropy weight term;

4. Stores prior experiences using a circular experience buffer. The agent updates the actor-critic using a mini-batch of experiences randomly sampled from the buffer.

The scheduler uses SACD algorithm to maximize the reward function (Section 3.3.4.4). The main goal of designing this approach is to automatically react to uncertainty, dynamism, and a variety of workloads without human intervention.

3.3.1. Algorithm optimization method

In SAC, the agent considers minimizing entropy (the average level of information (i.e., uncertainty) inherent in the variable's possible outcomes) besides the expected sum of rewards. In [26], the Shannon entropy (H) of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ and probability mass function $P(X)$ is defined as:

$$H(X) = -\sum_{(i=1)}^{n} P(x_i) log_b P(x_i) = E[-log_b P(X)], (1)$$

where b is the logarithm base. Common values for b are 2, Euler's number e, and 10. The corresponding units of entropy are the bits for $b = 2$, nats for $b = e$, and bans for $b = 10$.

One may also define the conditional entropy of two events X and Y taking values x_i and y_i respectively, as:

$$H(X|Y) = -\sum_{i,j} P(x_i, y_i) \log \frac{P(x_i, y_i)}{P(y_i)}, \quad (2)$$

where $P(x_i, y_i)$ is the probability that $X = x_i$ and $Y = y_i$. This quantity should be understood as the amount of randomness in the random variable X given the random variable Y.

Another important part of information theory is the concept of mutual information, which measures the mutual dependence between two variables. More specifically, it quantifies the information gain from a

random variable X about another random variable Y. It can also be represented as the decrease of disorder for a random variable Y on a random variable X [27]. The mutual information is defined by:

$$I(X;Y) = H(X) - H(X|Y). \tag{3}$$

Similar to conditional entropy, conditional mutual information measures the information contained in a random variable about another random variable, knowing the value of a third one. It can be written as below:

$$\begin{aligned} I(X;Y|S) &= H(X|S) - H(X|Y,S) \\ &= H(Y|S) - H(Y|X,S) \\ &= D_{KL}\left[p(x,y|s)||p(x|s)p(y|s)\right]. \end{aligned} \tag{4}$$

Haarnoja et al. [25] proposed a SAC algorithm whose policy aims to maximize the maximum entropy objective based on Eq. (5):

$$J(\pi) = \sum_{(t=0)}^{T} E_{(s_t,a_t)\sim\rho_\pi}\left[r_{(s_t,a_t)} + \alpha H(\pi(.|s_t))\right], \tag{5}$$

where $J(\pi)$ is the performance policy (π) between 0 to T and π is the agent's policy, T is the number of steps the agent takes during an epoch, the reward function is indicated by $r : S \times A \rightarrow R$, $\gamma \in [0,1]$ is the discount factor, $s_t \in S$ is the status of the environment at timestep t, $a_t \in A$ is the action that the agent takes at timestep t, and ρ_π is the trajectory distribution of the agent based on policy π. α is the temperature parameter, which shows the significant correlation between the entropy and the reward. Finally, the entropy of the policy π at the state s_t is $H(\pi(.|s_t)) = -\log\pi(.|s_t)$.

3.3.2. General policy iteration

To maximize the expected cumulative discounted reward, Howard [28] devised a standard approach that finds an optimal policy for every state called *policy iteration*. Policy iteration involves two steps: policy evaluation and policy improvement. Policy evaluation aims to find the accurate value function. In each step, the value function of the current policy updates to a new approximate value function. Thus, the policy improvement step (Eq. (7)) is performed repeatedly by applying Bellman's operator (i.e., the right side of Eq. (6)) [29].

$$[\tau_\pi V] = E_{a\sim\pi(.|S)}\left[r(S,a) + \gamma E_{S|S',a}\left[V(S')\right]\right], \tag{6}$$

$$[\tau_* V] = \max_\pi [\tau_\pi V]. \tag{7}$$

In Eqs. (6) and (7), τ_π represents the value iteration operator on policy π, and V is the initial value function that aims to converge to the optimal value function V^*. The convergence similarity of Bellman's operator and the optimality operator are considered based on their contraction mappings. Furthermore, the optimal policy π^* can be reached using the optimal value function, which minimizes a set of metrics between the current policy and the derived update policy with an initial policy π.

3.3.3. Soft policy iteration

By adding the reward term as the entropy of the policy, soft policy iteration extends the concept of general policy iteration. In particular, the agent aims to maximize both the environment's expected reward and the policy's entropy. Thus, Eq. (6) is regularized with an entropy term as shown below [29]:

$$\begin{aligned} \tau^\pi Q(s_t,a_t) &\triangleq r_{(s_t,a_t)} \\ &+ \gamma E_{a'\sim\pi}\left[Q(s_{t+1},a') - \log\pi(a'|s_t)\right]. \end{aligned} \tag{8}$$

Similar to the general policy and Eq. (6), τ^π indicates the soft value iteration operator, Q is a state-action function, r is the reward function of the state and action, γ is the discount factor representing the degree to which the future reward is affected by past actions, and a' is the next action. A_t is the policy improvement step, and the current Q function will be updated by the policy distribution ($J_\pi(\varphi)$) towards the softmax distribution. Specifically, this step helps us minimize the distance ("divergence") between the two distributions by minimizing their Kullback-Leibler (KL) divergence as below [24]:

$$\pi_{new} = arg\min_{\pi\in\prod}^{J_\pi(\varphi)}, \tag{9}$$

$$J_\pi(\varphi) = E_{s_t\sim D}\left[D_{KL}(\pi_\varphi(.|s_t))||\frac{\exp(\frac{1}{a}Q_\theta(s,.))}{Z_\theta(s_t)}\right]. \tag{10}$$

For the tabular case, Haarnoja et al. [30] have proved that this property guarantees the monotonic improvement of the policy. Later, Geist et al. [31] generalized the KL and entropy regularization to any formulation. Removing Z, which is a constant, and expressing the integration as an expectation, this equation reduces to the equation below:

$$\begin{aligned} J_\pi(\varphi) &= \\ & E_{s_t\sim D}\left[E_{a_t\sim\pi_\varphi}[\alpha\log(\pi_\varphi(a_t|s_t) - Q_\theta(s_t,a_t)]\right]. \end{aligned} \tag{11}$$

Finally, if the categorical distribution is used as the policy, the following equation is derived:

$$\begin{aligned} J_\pi(\varphi) &= \\ & E_{s_t\sim D}\Big[\pi_\varphi(.|s_t)^T\left[\alpha\log(\pi_\varphi(a_t|s_t) - Q_\theta(s_t,a_t)\right]\Big]. \end{aligned} \tag{12}$$

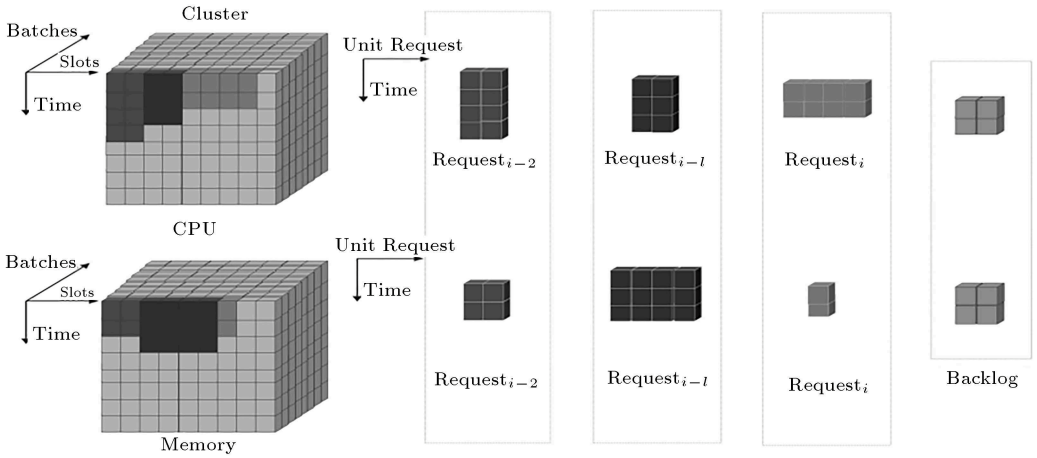

Figure 3. The scheduler input consisting of the data center's state and the jobs' profile in the queue and backlog.

3.3.4. The SACD core tasks

In this section, we will represent the SAC tasks based on [20]. These tasks consist of the states, actions, discount factor, and reward function.

1. *State* $s_t \in S$: State represents the status of the data center's resources s_t according to the cluster status, the queue of user requests at time t (incoming jobs in the queue), and backlog (jobs that cannot be placed in the scheduler's window) in a finite set. More precisely state (s_t) includes: (1) allocated and available data center resources, (2) the profile of each job in the requested queue, and (3) the backlog. Figure 3 shows an example of the state in a time step;

2. *Action* ($a_t \in A$): An action a_t selects a request (i.e., job) from the scheduler's window (waiting jobs). If the scheduler's queue has N slots for allocating the data center's resources at time t for the waiting jobs, then $a_t = \{a_t\}_1^N$ indicates the respective plan. Here, A is the action space of all possible job allocation strategies that the scheduler can select for the waiting jobs in its next iterations. It has $N + 1$ discrete actions ($\{\emptyset, 1, 2, ..., N\}$) in which $a_t = i(\forall i \in \{1..N\})$ denotes the ith job allocation in one of the N window slots [20]. Also, $a_t = \emptyset$ means no job will be allocated and the scheduler should do a MoveOn action (specifically, Mao et al. [5] proposed this action to reduce the action space of A);

3. *Discount factor* (γ): The parameter γ is a discount coefficient in the range of $[0, 1]$ that indicates the degree of impact of instant and future rewards on the reward function. This fixed parameter is manually defined by the designer. Due to delay in receiving the agent's reward and the consequences of long-term actions, we have used a value close to one for γ;

4. *Reward function* ($r \in R = S \times A \rightarrow (-\infty, 0)$): The actor's feedback in interaction with the data center and the incoming jobs leads the agent to receive a response from the scheduling environment. The actor seeks to maximize the reward function and minimize entropy. The reward function at time t in our scheduling problem is defined as below:

$$r_t = -1/T_j. \tag{13}$$

The parameter T_j is the alive time of a request (i.e., job) in the scheduler's window or backlog. The scheduler uses the expected total reward with a discount to evaluate its performance in an epoch. The time parameter T_j consists of the delay, hold, and dismiss times of the incoming jobs. The negative inverse ratio indicates the penalty for delay in processing existing jobs in the current state, keeping jobs in the new state of the data center, or losing a request due to the queue being full. The agent examines different policies and learns how to optimize its objectives.

5. *State-action function* ($Q(s_t, a_t)$): The effectiveness of various policies of the actor is evaluated by the state-action function. The critic iteratively estimates this function and corrects its predicted value to correct the actor's policy and improve it rapidly.

Below, we provide the algorithm for the discrete soft actor-critic-based scheduler. The algorithm tries to find the optimal policy based on the incoming jobs as well as the current state of the data center's resources.

3.3.5. Objective

In the proposed scheduling algorithm (Algorithm 1), the scheduler seeks to maximize the reward and entropy target functions. The entropy target function (as shown in the Eq. (14)) acts as a threshold to give an approximately equal chance for all possible and

Input: Status S, Policy parameterization $\pi_\varphi: S \rightarrow [0,1]^{|A|}$ (probability of data center states and scheduler actions)
Output: The scheduled sequence of requests [1...n] based on optimized parameters $\theta_1, \theta_2, \varphi$
Initialize online networks (Q) parameters: $Q_{\theta_1}: S \rightarrow \mathbb{R}^{|A|}$, $Q_{\theta_2}: S \rightarrow \mathbb{R}^{|A|}$
Initialize target networks ($\overline{Q}$)parameters: $\bar{Q}_{\theta_1}: S \rightarrow \mathbb{R}^{|A|}$,$\bar{Q}_{\theta_2}: S \rightarrow \mathbb{R}^{|A|}$
Initialize an empty replay buffer: $D \leftarrow \emptyset$ (action distribution)
Equalize weights of online and target network: $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$

Foreach iteration **do**
  Foreach jobs in the iteration **do**
    Sample action from the policy $a_t \sim \pi_\varphi(a_t|s_t)$
    Sample transition from the data center $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    Store the transition in the replay buffer $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$

  Foreach gradient step **do** (based on loss function and learning rate)
    Update Q-function parameters of online networks $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J(\theta_i); i \in \{1,2\}$
    Update policy weights of actor-network $\varphi \leftarrow \varphi - \lambda_\pi \hat{\nabla}_{\varphi_i} J_\pi(\varphi)$
    Update temperature (entropy) with Adam optimizer $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
    Update target network weights $\bar{Q}_i \leftarrow \tau Q_i + (1-\tau)\bar{Q}_i; i \in \{1,2\}$

Algorithm 1. The scheduler's algorithm based on discrete-soft-actor-critic.

available actions in the action space (The scheduler could choose non-frequent or rare selected actions in its exploration step).

$$J(\alpha) = \pi_t(s_t)^T \left[-\alpha \log \pi(s_t) + \bar{H}\right],$$

$$\bar{H} = 0.98 * \log |A|. \tag{14}$$

Additionally, entropy is an exploration tool that causes curiosity. It prevents the scheduler from getting stuck in local optimal points and making swift decisions without sufficient examination and searching.

3.3.6. Complexity analysis
In this section, we discuss the time and space complexity of the SACD algorithm. These are estimates, and the actual space complexity may vary depending on the implementation.

Time complexity
The time complexity of the algorithm in formal notations is:

$$O(T * P + T * N * D * A + T * A * (D + 1)),$$

*This time complexity assumes that the NN takes $O(1)$ to evaluate a single state-action pair.

Where T is the number of training iterations; P the number of parameters in the NN; N the size of the replay buffer; D the dimensionality of the state space; A the number of discrete actions.

The first term, $T * P$ represents the time complexity of updating the NNs, which is proportional to the number of parameters in the networks. The second term, $T * N * D * A$, represents the time complexity of sampling from the replay buffer and computing the Q-values for each action. It involves retrieving a batch of transitions from the replay buffer and computing the Q-value for each state-action pair. The time complexity of this operation is proportional to the size of the replay buffer, the dimensionality of the state space, and the number of discrete actions.

The third term, $T * A * (D+1)$, represents the time complexity of computing the policy distribution and the entropy regularization term. It involves the logits computation for each action, the softmax probabilities, and the entropy regularization. The time complexity of this operation is proportional to the number of discrete actions and the state space dimensionality.

Space complexity
The SACD algorithm's space complexity is $O(P + N * (D + 1 + A))$ where p is the number of parameters in the NNs used in the algorithm, N is the size of the replay buffer, D is the dimensionality of the state space of the problem being solved by the algorithm, and A is the number of discrete actions in the action space of the problem being solved. The space complexity of the NN itself can be expressed as $O(P)$, and the space complexity of the replay buffer can be denoted by $O(N * (D + 1 + A))$. The total space complexity of the algorithm is the sum of these two terms.

4. Evaluations

The performance of the approach is evaluated with respect to the balance of slowdown and resource (i.e., CPU and memory) utilization using Google's dataset [32].

4.1. Environmental setup
The evaluation and implementation of the approach are based on DeepRM. The data center has a centralized M cluster in which each cluster has K resources. It is assumed that each cluster has the same type and quantity of resources. The clusters' resource fragmentation is not considered and each job has the same execution time across all clusters. Resource allocation is non-preemptive. The queue and backlog have fixed sizes. We used Python 3 as the programming language. The self-adaptive algorithm is modeled by the Numpy framework and PyTorch [33] library. The hardware used for evaluation has 8 virtual CPU cores, 16 GB of RAM, and 30 GB of disk

Table 2. The initial discrete-soft-actor-critic algorithm parameters used to train the scheduler.

Parameter	Value
Critic's learning rate	0.01
Number of epochs	500
Simulation length	50
Number of sequences	10
Maximum episode length	2000
Number of data center resources	2
Number of allowed jobs in the scheduling window	10
Time horizon	20
Maximum job duration	15
Maximum requested resource slot	10
Maximum available resource slot	10
Backlog size	60
New job rate	0.5
γ	0.99
α	0.003
Target_entropy_ratio	0.98
Target_critic update_interval	250
Starting_steps (random)	80000
Memory size	100000
Batch size	64
Initial weight	$\mathcal{N}\left(\mu, \sigma^2\right), \mu = 0, \sigma^2 = 0.1$
Initial bias	0

Table 3. The normalized real-world dataset.

Normalized service time	Normalized requested CPU of each job	Normalized requested Memory of each job
0.722512039	0.007244731	0.131669952
0.00682183	0.01473928	0.132658107
0.675561798	0.018781795	0.144515817
0.301163724	0.007063045	0.084115618

space. The experiments are executed on a Windows 10 and compiled by PyCharm [34].

4.2. Real-world dataset

The approach's performance was evaluated using Google's Brog cluster dataset. We calculated the difference between the start_time and the end_time of the measurement period as the job service time and the CPU and memory usage for each job based on maximum_usage. Finally, we normalized the job service time, CPU, and memory usage based on the following formula:

$$Z = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{15}$$

where Z is the normalized job service time and converted to the interval of [0, maximum duration/maximum requested] resource slot (shown in Table 2). Table 3 shows an example of the data used in this test.

4.3. Evaluation assumptions

In the performed evaluations, there exist 16 data center nodes, with the range of resource of each job profile being between 0 to 15. The incoming jobs' queue capacity is 10, and the backlog capacity is 60. The simulation length is equal to the number of jobs * num_exe. We have used two types of jobs: (1) dominant and (2) usual. The dominant jobs request resource(s) in the interval $[0.05r, 0.025r]$ of the uniform distribution, and usual jobs announce their requests in the interval $[0.005r, 0.01r]$. The job request time corresponds to

Table 4. Performance comparison of the proposed approach with the random, Tetris, and Shortest-job-first algorithms.

Type	Random	Tetris	Shortest-job-first	SACD
CPU utilization	0.258673	0.092534	0.075208	0.261788
Memory utilization	0.263191	0.092712	0.078417	0.279105
Balance of slowdown	6.562134	7.813635	7.072100	6.588019
Slowdown	10.362481	11.64630	8.342642	10.216499

Table 5. Performance comparison of the proposed approach with the DeepRM and DeepScheduler.

Type	DeepRM	DeepScheduler	SACD
CPU utilization	0.133744	0.06455	0.27526
Memory utilization	0.120993	0.06078	0.27124
Balance with the slowdown	20.36493	18.86114	11.99066
Slowdown	11.39213	14.14282	6.26761

the uniform distribution and the patterns introduced in Section 5.2. The maximum requested resources (r) and the time interval of resource usage (t) are 10 and 15, respectively. Finally, the proposed approach was evaluated using Google's 2019 dataset [35]. Jobs arrive online and not simultaneously at the data center based on the Bernoulli process. The incoming job rate is kept constant during the training and evaluation processes.

4.4. Evaluation metrics

The convergence speed of the RL approach is not applicable and informative for cloud applications such as scheduling [20]. Therefore, we have used Liang et al.'s metrics [20] for a better assessment of the approach's performance.

Based on [20], if a set of jobs $J = \{j_1, ..., j_i, ..., j_N\}$ is assumed in which j_i consists of arrival time (t_i^a), finish time (t_i^f), and execution time (t_i^e), then the average job slowdown and the waiting time would be defined as $S_{avg} = \frac{1}{N}\sum_{i=1}^{n}\frac{t_i^f - t_i^a}{t_i^e} = \frac{1}{n}\sum_{i=1}^{N}\frac{c_i}{T_i}$, and $t_{wi} = t_i^s - t_i^a$, respectively.

The balance of slowdown is another metric that consists of the maximum, minimum, and average slowdowns. It shows how well the scheduler can distribute jobs on the data center resources. The balance of slowdown is defined as: Degree of Balance $(DB) = \frac{S_{\max} - S_{\min}}{S_{avg}}$ where $S_{\max} = \max_1^n\{\frac{c_i}{T_i}\}$ and $S_{\max} = \min_1^n\{\frac{c_i}{T_i}\}$.

4.5. The results

Based on Table 2, we examined the scheduler's trained policy and the performance of the approach. Then, we compared the approach's performance with DeepRM and DeepScheduler, which are based on the RL algorithm. Finally, we studied the impact of the learning parameters of the scheduling algorithm.

4.6. Scheduler comparison

The experimental results indicate that the scheduling algorithm (Algorithm 1) improves the evaluated criteria compared with the Tetris and random scheduling algorithms. The SACD algorithm focuses on fair job response time with the proper balance with the slowdown and better utilization of data center resources, while the shortest-job-first policy only schedules jobs with minimum service times. Table 4 demonstrates the performance of the proposed approach compared to the classical and heuristic-based algorithms.

4.7. Approach comparison

Table 5 shows the comparison results of the proposed approach. The approach outperforms DeepRM and DeepScheduler without any manual reconfiguration.

4.7.1. The state-action critic function

Figures 4 and 5 show the critic's prediction error and target critic (Q_1, Q_2) in the learning phase. The horizontal axis represents the training iteration, while the vertical axis represents the loss function of the state-action. Initially, the critic makes completely random predictions and the loss function grows quickly. After a few iterations, the scheduler, unaware of

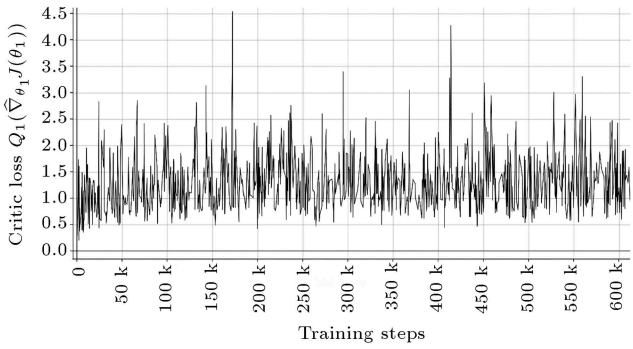

Figure 4. The online critic's prediction error for the state-action function.

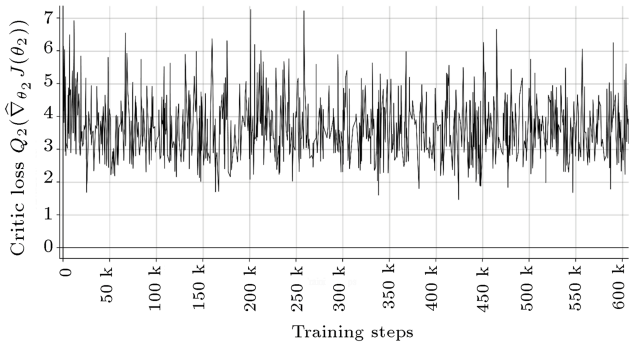

Figure 5. The target critic's prediction error for the state-action function.

its performance, learns that its predictions are not compatible with the target state. It tries to improve its experiences through trial-and-error interactions in the environment. This process helps the scheduler to converge to the assumed objective policy.

$$Q\_loss = MSE(q_{pred}, q_{target}), \tag{16}$$

$$q_{pred} = Q_{network}(state, action), \tag{17}$$

$$q_{target} = reward + \gamma * (targetQ(s', a')) - \alpha * \log \pi(a'|s'), \tag{18}$$

$$J_Q(\theta) = E_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_Q(s_t, a_t) - q_t)^2\right], \tag{19}$$

where D is all the transition data in a replay buffer, Q_Q is q_{pred}, and q_t is q_{target}.

4.7.2. The actor policy

The actor can choose a job from the queue based on the action probability generated by the policy π. π is a mapping from the state s_t to action a_t, which generates a probability for each possible action. For instance, given the action probability $P = \{p_1, ..., p_N\}$ for N actions, p_i denotes the probability that action a_i will be selected. Exploration is allowed in this research. The policy is estimated by a NN $\pi(a|s, \theta)$, where a is an action, s is the state of the system, and θ is the weight of the policy network.

Figure 6 shows how the actor automatically optimizes its policy. The actor updates the policy parameters in the direction suggested by the critic based on Eq. (21). The actor's network is updated by minimizing the actor's loss policy in Eq. (20). The policy loss function leads the scheduler to the optimal policy. Initially, the parameters in the NN are random, resulting in a random policy. The actor initially makes completely random decisions and the pending job queue quickly grows. After a few iterations in which the policy loss is decreased, the actor understands that its scheduling policy does not match the target state

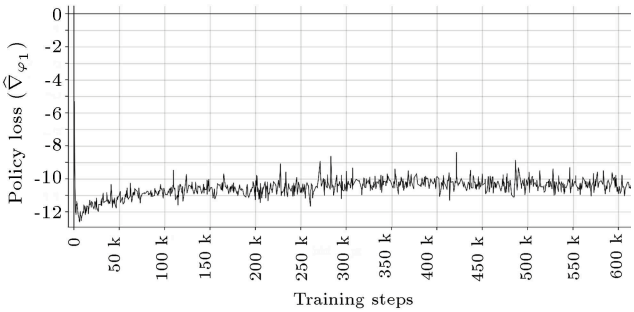

Figure 6. The actor policy error based on the assumed objectives.

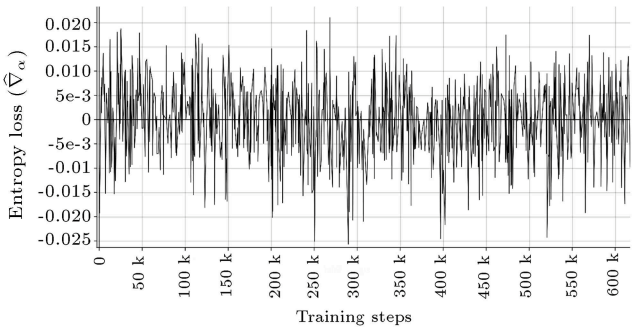

Figure 7. The entropy changes during the scheduler's training process.

of the environment. Thus, it tries to change its actions based on the criticism score of the critic. This process is a non-trivial step, and during the training process, the scheduler changes its policy to reach the given objective. It will also be aware of its environmental changes, especially incoming jobs.

$$policy\_loss = -J_\pi(\varphi), \tag{20}$$

$$J_\pi(\varphi) = E_{s_t\sim D}\left[\pi_t(s_t)^T[\alpha \log \pi_\varphi(s_t) - Q_\theta(s_t)]\right]. \tag{21}$$

4.7.3. Target entropy

The parameter (α) is responsible for improving or correcting the entropy of the scheduler's policy. Eq. (14) and Figure 7 demonstrate the entropy changes during the learning phase. Target entropy acts as a threshold. Entropy loss increases α and the entropy. When the entropy is lower or higher than the target entropy, the value of α is decreased. Therefore, by setting the target entropy ratio, we are setting the exploration/exploitation ratio in the scheduler.

5. Practical limitations

Despite the advantages offered by the approach in solving automatic job scheduling and resource management, it also has its disadvantages and practical limitations, which are described below:

1. *Space and time overhead*: The scheduler learns to

converge based on a NN method that lacks prior knowledge about its estimations. Thus, it must iteratively test and evaluate its actions to improve its policy. On the other hand, entropy causes turmoil to select actions. So, the scheduler needs more experience and time to learn a stable policy based on the proposed training set;

2. *Lack of resource heterogeneity in the data center*: Data centers consist of various resources with different capacities. The variation in data centers' resources isn't considered in this research. This decision is for simplification purposes and may be a limiting factor in practical applications;
3. *Non-preemptive job scheduling*: In this type of scheduling, jobs exclusively use resources until their execution is finished. This approach results in less efficient resource utilization and longer waiting and response times compared to preemptive scheduling. This assumption is made for simplification purposes;
4. *Independent incoming jobs*: Each batch job process consists of a group of tasks with a Directed Acyclic Graph (DAG) structure, which wasn't considered in the proposed approach due to the relaxing assumption of independent incoming jobs. Additionally, the job's (i.e., Virtual Machine (VM) or container application) location on the hosts (i.e., servers) in the data center has a side effect on the data center's resource utilization.

6. Conclusions

In this paper, we proposed a resource management approach for cloud computing environments. The scheduler automatically and without prior knowledge allocates data center resources to users' submitted jobs. It schedules incoming jobs based on an Reinforcement Learning (RL) algorithm. After several iterations, the scheduler learns how to interact with the environment to accomplish the given objectives. This characteristic of the algorithm gives the scheduler the ability to adapt to incoming jobs and properly execute them in the data center.

Although progress has been achieved in applying the proposed adaptive approach, there are still some recommendations and improvements that can be worked on to enhance the performance even further. The following is a summarized list of future work that can be performed in this regard:

1. In the Discrete-Soft-Actor-Critic (SACD) algorithm, it is possible to use parallel actors. This feature can help the scheduler gain more trajectories without experiencing a new state. Having multiple actors helps correlate the encountered states and attenuate the feedback loops while allowing us to leverage the parallel architecture of modern CPUs and GPUs;
2. As a future direction to reduce learning time, further research can be performed on optimizing learning hyperparameters such as the number of neurons, epochs, and minibatch size using typical optimization heuristics [36];
3. The host temperature problem in data centers was outside the scope of the current research. By using predictive machine learning-based scheduling algorithms that manage data centers' energy consumption by monitoring their sensor data, this aspect can be also taken into account [23];
4. Based on the limitations mentioned in Section 6, non-homogeneous resources and multi-hybrid cloud infrastructures can be considered in future extensions to formulate resource provisioning problems.

Compliance with ethical standards

This study has received no funding from any organization.

Conflict of interest

All of the authors declare that they have no conflict of interest.

Ethical approval

This article doesn't contain any studies with human participants or animals performed by any of the authors.

Funding

No funding was received for this research.

Availability of data and materials

Not applicable as already public datasets are used in this research.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

The first author participated in the design, implementation, and evaluation phases of this research as well as creating the first manuscript draft. The second author participated in the design, verification, and revising of the original draft.

References

1. Rjoub, G., Bentahar, J., Abdel Wahab, O., et al. "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems", *Concurrency and Computation: Practice and Experience*, (2020). DOI: 10.1002/cpe.5919.
2. Maqableh, M., Karajeh, H., and Masa'deh, R. "Job scheduling for cloud computing using neural networks", *Communications and Network*, **06**(03), pp. 191–200 (2014). DOI: 10.4236/cn.2014.63021.
3. Singh, S. and Chana, I. "QoS-Aware autonomic resource management in cloud computing", *ACM Computing Surveys*, **48**(3), pp. 1–46 (2016). DOI: 10.1145/2843889.
4. Liang, S., Yang, Z., Jin, F., et al. "Data centers job scheduling with deep reinforcement learning", In *Proceedings of 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Singapore*, Singapore, pp. 906–917 (2020).
5. Mao, H., Alizadeh, M., Menache, I., et al. "Resource management with deep reinforcement learning", In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, Atlanta GA, USA, pp. 50–56 November-(2016).
6. Haarnoja, T., Zhou, A., Abbeel, P., et al. "Soft actor-critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor", arXiv preprint arXiv: 1801.01290 (2018).
7. "Apache Hadoop 3.3.0-Hadoop: Fair Scheduler", URL: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html Access date: 7 September (2020).
8. Ghodsi, A., Zaharia, M., Hindman, B., et al. "Dominant resource fairness: Fair allocation of multiple resource types", In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, Boston, MA, USA, pp. 323–336 (2011).
9. Song, W., Xiao, Z., Chen, Q., et al. "Adaptive resource provisioning for the cloud using online bin packing", *IEEE Transactions on Computers*, **63**(11), pp. 2647–2660 (2014). DOI: 10.1109/tc.2013.148.
10. Grandl, R., Ananthanarayanan, G., Kandula, S., et al. "Multi-resource packing for cluster schedulers", In *Proceedings of the 2014 ACM Conference on SIGCOMM*, Chicago Illinois, USA, pp. 455–466 (2014).
11. Pezzella, F., Morganti, G., and Ciaschetti, G. "A genetic algorithm for the flexible job-shop scheduling problem", *Computers & Operations Research*, **35**(10), pp. 3202–3212 (2008). DOI: 10.1016/j.cor.2007.02.014.
12. Azad, P. and Navimipour, N. "An energy-aware task scheduling in the cloud computing using a hybrid cultural and ant colony optimization algorithm", *International Journal of Cloud Applications and Computing*, **7**(4), pp. 20–40 (2017). DOI: 10.4018/ijcac.2017100102.
13. Huang, J., Xiao, C., and Wu, W. "RLSK: A job scheduler for federated kubernetes clusters based on reinforcement learning", In *Proceedings of 2020 IEEE International Conference on Cloud Engineering (IC2E)*, Sydney, Australia, Australia, (2020).
14. Mao, H., Schwarzkopf, M., Venkatakrishnan, S., et al. "Learning scheduling algorithms for data processing clusters", In *Proceedings of the ACM Special Interest Group on Data Communication*, Beijing, China (2019). DOI: 10.1145/3341302.3342080.
15. Chen, W., Xu, Y., and Wu, X. "Deep reinforcement learning for multi-resource multi-machine job scheduling", arXiv preprint arXiv:1711.07440 (2017).
16. Ye, Y., Ren, X., Wang, J., et al. "A new approach for resource scheduling with deep reinforcement learning", arXiv preprint arXiv:1806.08122 (2018).
17. Domeniconi, G., Lee, E., Venkataswamy, V., et al. "CuSH: cognitive scheduler for heterogeneous high-performance computing system", In *Proceedings of DRL4KDD 19: Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD)*, Alaska, USA (2019).
18. Cheong, M., Lee, H., Yeom, I., et al. "SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster", *IEEE Access*, **7**, pp. 153432–153444 (2019). DOI: 10.1109/access.2019.2948150.
19. Li, F. and Hu, B. "DeepJS: Job scheduling based on deep reinforcement learning in cloud data center", In *Proceedings of the 2019 4th International Conference on Big Data and Computing*, Guangzhou, China, pp. 48–53 (2019).
20. Liang, S., Yang, Z., Jin, F., et al. "Data centers job scheduling with deep reinforcement learning", In *Proceedings of 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Singapore, Singapore, pp. 906–917 (2020).
21. Guo, W., Tian, W., Ye, Y., et al. "Cloud resource scheduling with deep reinforcement learning and imitation learning", IEEE, *Internet of Things Journal*, **8**(5), pp. 3576–3586 (2021).
22. Xu, M., Song, C., Ilager, S., et al. "CoScal: Multifaceted scaling of microservices with reinforcement learning", IEEE *Transactions on Network and Service Management*, **19**(4), pp. 3995–4009 (2022).
23. Zhong, Z., Xu, M., Rodriguez, M.A., et al. "Machine learning-Based orchestration of containers: A taxonomy and future directions", *ACM Computing Surveys*, **54**(10)s, pp. 1–35 (2022).
24. Christodoulou, P. "Soft actor-critic for discrete action settings", arXiv preprint arXiv: 1910.07207 (2019).
25. Haarnoja, T., Zhou, A., Abbeel, P., et al. "Soft actor-critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor", arXiv preprint arXiv: 1801.01290 (2018).
26. Arndt, C., *Information Measures*, 1st ed. Berlin: Springer (2004).

27. Aubret, A., Matignon, L., and Hassas, S. "A survey on intrinsic motivation in reinforcement learning ", arXiv preprint arXiv:1908.06976 (2019).

28. Howard, R. "Dynamic programming and Markov processes", Cambridge: M.I.T. Press, (1972).

29. "In-depth review of soft actor-critic", URL: https://towardsdatascience.com/in-depth-review-of-soft-actor-critic-91448aba63d4, Access date: May 17th, (2021).

30. Haarnoja, T., Zhou, A., Hartikainen, K., et al. "Soft actor-critic algorithms and applications", arXiv preprint arXiv:1812.05905 (2019).

31. Geist, M., Scherrer, B., and Pietquin, O. "A theory of regularized Markov decision processes", arXiv preprint arXiv:1901.11275 (2019).

32. "Google's dataset", URL: gs://clusterdata_2019_a/ instance_usage-0000001.json.gz.

33. "PyTorch", URL: https://pytorch.org/, Access date: March 12th, (2020).

34. "PyCharm", URL: https://www.jetbrains.com/ pycharm/, Access date: March 14th, (2020).

35. "Google Cluster Workload Traces 2019", URL: https://research.google/tools/datasets/google-cluster-workload-traces-2019/, Access date: April 1th, (2020).

36. Khan, T., Tian, W., Zhou, G., et al. "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," *Journal of Network and Computer Applications*, **204**, pp. 1–51 (2022).

Biographies

Ahmadreza Sheibanirad received his BSc degree in Software Engineering from the University of Isfahan (2016), Isfahan, Iran, and his MSc degree in Software Engineering from Iran University of Science and Technology (2020), Tehran, Iran. His main research interests include cloud computing, automation, and applications of self-adaption in the domain of computer science.

Mehrdad Ashtiani received his BSc degree in Software Engineering from Iran University of Science and Technology (2009), Tehran, Iran, and his MSc degree in Software Engineering from the same institution in 2011, and his PhD degree from Iran University of Science and Technology (2015), Tehran, Iran. His main research interests include trust modeling and the applications of uncertainty modeling in the domain of computer science.