This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

Joint Resource Overbooking and Container Scheduling in Edge Computing

Zhiqing Tang, *Member, IEEE,* Fangyi Mou, Jiong Lou, *Member, IEEE,* Weijia Jia, *Fellow, IEEE,* Yuan Wu, *Senior Member, IEEE,* and Wei Zhao, *Fellow, IEEE*

Abstract—Containers have gained popularity in Edge Computing (EC) networks due to their lightweight and flexible deployment advantage. In resource-constrained EC environments, overbooking container resources can substantially improve resource utilization. However, existing work overlooks the complex interplay between resource provisioning and container scheduling, which may result in performance degradation or inefficient resource utilization due to highly dynamic resource heterogeneity in EC. To address this issue, this paper presents a novel joint Resource Overbooking and Container Scheduling (ROCS) algorithm. Our approach accounts for resource heterogeneity and the geographical distribution of edge nodes, and we formulate the ROCS problem to consolidate various costs and revenues into a single profit metric for service providers. To enhance resource utilization and maximize the profit of the service providers, we develop an efficient algorithm that operates within a hybrid action space scheme by leveraging soft actor-critic reinforcement learning. Furthermore, we introduce a risk assessment mechanism to mitigate overbooking risks. Large-scale simulations with real-world data traces demonstrate the efficacy of our proposed ROCS algorithm, validating its advantage of improving resource utilization within EC networks.

Index Terms—Resource overbooking, container scheduling, edge computing, soft actor-critic reinforcement learning

1 Introduction

In Edge Computing (EC), edge nodes with limited computing resources are positioned closer to users at the network edge, effectively supplementing mobile users' computing resources and providing lower latency compared to cloud data centers [1]. Additionally, containers have become prevalent in EC for deploying applications to handle user tasks due to their lightweight and easy-to-deploy features [2], [3]. To manage large-scale edge nodes and container clusters, various platforms, such as Kubernetes and KubeEdge, have been developed [4], [5].

When deploying containers in edge nodes, resources are managed based on requests rather than actual usage [5]. However, many containers do not fully utilize requested resources, leading to significant resource waste. For example, studies of Google's production clusters reveal disparities of about 53% for CPU and 40% for memory [6]. Actual CPU usage ranges between 20%-35%, and memory usage between 20%-40% [7]. Due to Kubernetes scheduling limitations, these resources cannot be reallocated even if they are under-utilized, which is particularly unfavorable in EC [8]. Resource utilization can be improved through overbooking [7], [9], allowing containers to execute using idle resources from existing containers. This technique has been exploited in cloud data centers at various levels, e.g., the kernel [10], the hypervisor [11], and the container cluster scheduler. For example, OpenShift [12] and Mesos [13] schedulers perform overbooking decisions with a static threshold. Besides, researchers have also proposed effective overbooking schemes for cloud data centers [7], [14]–[17]. In EC, overbooking is in its preliminary stage, with several current studies focusing on designing overbooking methods like auction mechanisms [18], [19] and pricing models [20], [21].

It is a complicated yet technically challenging task to determine different overbooking thresholds for each edge node at every moment while considering the resource heterogeneity, geographical distribution of edge nodes, and the impact of different types of tasks. Existing studies often establish a static threshold for resource usage and then overbook resources as many as possible within this constraint [7], [14], [16]. Researchers have attempted to overbook resources dynamically by predicting resource usage [20]. However, these studies have largely overlooked that the heterogeneity and geographical distribution of edge nodes can result in increased overbooking risks with an ill-suited threshold. For example, if a static threshold is set as in typical overbooking schemes [16], then nodes with fewer computing resources face a significantly higher overbook-

- *Zhiqing Tang is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, and also with the State Key Lab of Internet of Things for Smart City, University of Macau, SAR Macau, China, and also with Yunnan Key Laboratory of Software Engineering, Kunming 650091, Yunnan, China. (E-mail: zhiqingtang@bnu.edu.cn)*
- *Fangyi Mou is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China. (E-mail: moufangyi@uic.edu.cn)*
- *Jiong Lou is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China, and also with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China. E-mail: (lj1994@sjtu.edu.cn)*
- *Weijia Jia is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, and also with Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai 519087, China. (E-mail: jiawj@bnu.edu.cn)*
- *Yuan Wu is with the State Key Lab of Internet of Things for Smart City, University of Macau, SAR Macau, China. (E-mail: yuanwu@um.edu.mo)*
- *Wei Zhao is with CAS Shenzhen Institute of Advanced Technology, Shenzhen 518055, China. (E-mail: zhao.wei@siat.ac.cn)*

(Corresponding authors: Weijia Jia and Yuan Wu.)

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

ing risk. Moreover, due to the geographical distribution of edge nodes, different nodes may receive burst resource requests at varying times, such as sudden traffic accidents at specific locations [20], [22]. Inappropriate thresholds may exacerbate overbooking risks in these cases. Moreover, a too-low threshold fails to utilize resources fully. As a result, a key challenge lies in dynamically adjusting the overbooking threshold for each edge node to utilize resources while minimizing the overbooking risks optimally.

Furthermore, exploring how to jointly make resource overbooking and container scheduling decisions with a long-term view presents a significant challenge. From a global and long-term perspective in EC, the relationship between resource overbooking and container scheduling is intricate, given the trade-off between risk (e.g., the number of container evictions) and resource utilization. On the one hand, dynamic thresholds effectively reduce overbooking risks, such as the number of container evictions. However, they do not globally optimize task completion time. On the other hand, existing container scheduling studies make decisions based on various resource constraints (e.g., CPU, memory, and storage) [3], [23], [24]. However, these constraints may no longer be valid when resources are overbooked, significantly increasing risks. For instance, when the number of requests rises in a specific area in EC, or running containers request more resources during peak hours [20], [25], the threshold should be lowered to avoid scheduling additional containers to these nodes, thereby reducing risk. Furthermore, container scheduling influences edge node resource utilization, affecting the adjustment of dynamic overbooking thresholds for these nodes.

These challenges are not easily addressed by designing heuristic or auction algorithms [18], [20]. In contrast, Reinforcement Learning (RL) algorithms can fully consider the impact of continuous decisions [26]. The heterogeneity of EC and long-term benefits can be considered through state and reward functions. Moreover, joint resource overbooking and container scheduling decisions can be implemented by modifying the RL action space to a hybrid action space [27]. Therefore, RL-based algorithms are well-suited for decision-making in EC. A soft actor-critic-based RL algorithm is proposed to jointly make efficient resource overbooking and container scheduling decisions.

To the best of our knowledge, there exist few studies investigating the joint Resource Overbooking and Container Scheduling (ROCS) problem in EC to maximize the overall profit for the service provider. We first extract features such as resource usage and task requests and consider the resource heterogeneity and geographical distribution of edge nodes. Then, the ROCS problem is modeled as a profit maximization problem that considers both the cost and revenue of containers. To tackle the abovementioned challenges, we propose a ROCS algorithm based on the soft actor-critic algorithm [28] and design a hybrid action space to determine the overbooking threshold and jointly make container scheduling decisions. The reward function allows for proper consideration of long-term profit. Lastly, we design a risk check mechanism to reduce the overbooking risk by evaluating the scheduling decisions and evictions. We conduct experiments with a large-scale real-world dataset from Alibaba [8] to demonstrate the effectiveness of our algorithm.

The contributions of this paper can be summarized as follows.

1) We formulate the joint resource overbooking and container scheduling problem in a heterogeneous EC environment, considering resource limitations and heterogeneity while unifying different costs into the profit of the service provider.
2) We propose the ROCS algorithm based on soft actor-critic RL with a hybrid action space to adjust the overbooking threshold and make container scheduling decisions jointly. Furthermore, a risk check mechanism is designed to reduce the overbooking risk further.
3) Our algorithm is assessed based on a large-scale real-world data trace. Compared to the baseline approaches, our ROCS algorithm achieves up to a 75% increase in profit with varying task numbers while significantly reducing the overbooking risk and enhancing resource utilization.

The remainder of the paper is organized as follows. In Section 3, we describe the system model and problem formulation. Next, the ROCS algorithm is proposed in Section 4. Then, performance evaluation is presented in Section 5 and some issues are discussed in Section 6. Finally, we conclude the paper and discuss future directions in Section 7.

2 Related Work and Motivation

2.1 Container Scheduling in Edge Computing

Containers have been widely used in EC to deploy various applications because of their lightweight and easy-to-deploy features. Pan *et al.* [3] study the retention-aware container caching problem in serverless EC and propose an online algorithm to reduce the overall cost. Hu *et al.* [2] propose a containerized edge computing framework for dynamic resource provisioning. They also study the joint service request scheduling and container retention problem and propose an online co-decision scheme to minimize the long-term system cost [29]. Gu *et al.* [24] study a layer-aware microservice placement and request scheduling at the edge to increase the throughput and number of hosted microservices. Tang *et al.* [4] propose a layer-aware container scheduling algorithm based on policy optimization to reduce the deployment cost in EC. Container migration problems in EC have also been studied [30].

However, due to the nature of containers, resources that have already been allocated cannot be reallocated even if they are not used. Therefore, none of the above studies have well considered the resource overbooking problem during container scheduling. To solve this issue, Trimaran [31] is proposed to make the scheduler aware of the gap between resource allocation and actual resource utilization, which is a collection of load-aware scheduler plugins. However, Trimaran can only set static overbooking thresholds and only considers CPU usage, not other resources such as memory, hard disk, etc., so it can still cause many problems.

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

2.2 Resource Overbooking in Edge Computing

Resource overbooking has been widely used in cloud computing while still in its infancy in EC. In cloud computing, various schedulers [10]–[13] and researchers [7], [14]–[17] have considered resource overselling and proposed different solutions. In EC, Zhang *et al.* [21] propose a novel pricing-based dynamic resource allocation model through an overbooking mechanism to overbook as many idle resources as possible with high QoS satisfaction. Tang *et al.* [20] propose an overbooking mechanism, including a cancellation policy and a resource prediction method to meet the needs of different users in EC. Liwang *et al.* [18] introduce a novel computing resource provisioning mechanism empowered by overbooking, which is formulated as a multi-objective optimization problem that aims to maximize the expected utilities of end-users, edge, and cloud. Besides, they also adopt overbooking and propose a hybrid market unifying futures and spot to facilitate resource trading among an edge server and multiple smart devices [19]. Zanzi *et al.* [32] propose an end-to-end network slicing orchestration solution that allows requesting network slices on-demand and displays the achieved multiplexing gain through overbooking. Gao *et al.* [33] overbook the backup virtual machine to improve resource utilization in mobile edge computing effectively.

All of the above researches set an overbooking threshold and overbook the resources according to the threshold. However, they all set a static threshold, and none of these studies considers the dynamic adjustment of overbooking thresholds or the interplay between resource overbooking and container scheduling. To better illustrate the motivation of this paper, we give an example of motivation as follows.

2.3 Motivation Example

Tasks are scheduled to edge nodes and processed by different containers. Service providers generate revenue by offering limited edge node resources. To maximize the revenue, resources can be overbooked [20]. To illustrate more clearly the interplay between resource overbooking and container scheduling, we present the flow of the ROCS problem in Fig. 1. The different colors in Fig. 1 represent different tasks. And different shapes in Fig. 1 represent task usage, task request, and task eviction, respectively. a_t^c and a_t^r represent the container scheduling decision and the overbooking threshold of the corresponding node, respectively. Starting from time $t = 2$, in chronological order, we draw the resource usage of the selected node (i.e., the node where the task is scheduled) at each time in turn. For example, when $t = 2$, $a_2^c = 1$, indicating that the current task is dispatched to node 1, so we draw the resource usage of node 1. Besides, $a_2^r = 0$ indicates that no resources are overbooked when $t = 2$.

Subsequently, when $t = 7$, $a_7^r = 0.6$, signifying that resources are overbooked. When $t = 8$, the total resource utilization of node 5 is already at a relatively high level. Thus, scheduling to node 5 may not be a good choice at this point. We need to either adjust the scheduling of the container or adjust the resource overbooking threshold of node 5 to avoid possible resource over-utilization situations. When $t = 9$, trade-off between container scheduling and node resource overbooking threshold is not good, resulting in container eviction since there are not enough available resources. As a result, container scheduling and resource overbooking need to be jointly considered to use as many idle resources as possible while avoiding container evictions as much as possible. In this paper, we aim to make joint decisions on container scheduling and overbooking thresholds to maximize benefits.

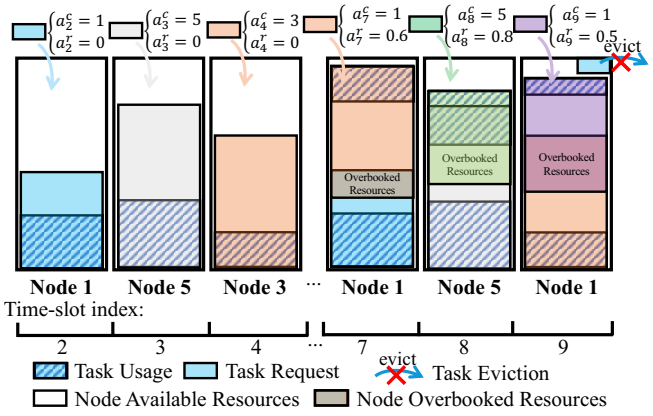


Fig. 1. Overview of the ROCS problem

3 System Model and Problem Formulation

In this section, we first introduce the EC system model in Section 3.1. Next, we define the cost in Section 3.2. Finally, we formulate and analyze the problem in Section 3.3.

3.1 System Model

We define edge node, container, and task as follows. For ease of reference, the main notations used in this paper are summarized in TABLE 1.

Edge node: A set of edge nodes, represented by $\mathbf{N} = \{n_1, n_2, ..., n_{|\mathbf{N}|}\}$, is deployed at the network edge to provide computing resources. For each node $n \in \mathbf{N}$, CPU and memory capacities are denoted as c_n and m_n, respectively. At time t, CPU and memory usages are denoted as $c_n^u(t)$ and $m_n^u(t)$, respectively. Each node maintains a running container list $\mathbf{P}_n(t)$. Furthermore, l_n represents the location of node n.

Container: A container is the smallest unit in which a task runs. The container set is denoted as $\mathbf{P} = \{p_1, p_2, ..., p_{|\mathbf{P}|}\}$. Each container $p \in \mathbf{P}$ has a requested CPU c_p^r and requested memory m_p^r. Real-time CPU and memory usage are denoted as $c_p^u(t)$ and $m_p^u(t)$, respectively. Generally, a container's resource requests match the requests of the task running within it. For example, if container p is created to process task k, resource requests are set to match task k's requests.

Task: A set of tasks, represented by $\mathbf{K} = \{k_1, k_2, ..., k_{|\mathbf{K}|}\}$, is generated by users and scheduled to various edge nodes for processing. For each task $k \in \mathbf{K}$, the arrival time and expected finish time are denoted as t_k and d_k, respectively. The data size is z_k. CPU and memory requests are denoted as c_k^r and m_k^r, respectively. Upon task k completion, a value v_k is obtained, which is inversely proportional to completion time if finished without eviction;

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

otherwise, the value is 0 [34]. Moreover, each task has a QoS level $o_k \in \{o^G, o^B\}$, divided into two types: Guaranteed o^G and Best Effort o^B [17]. To manage overbooking effectively, the CPU management strategy adopts a static strategy [5], where CPU resources requested by Guaranteed tasks cannot be overbooked. For Best Effort tasks, shared CPUs are allocated and can be overbooked.

To manage overbooking more properly, the CPU management strategy is set to static strategy [35]. Specifically, the dedicated CPUs are allocated for Guaranteed tasks (i.e., the allocated containers). Moreover, for other tasks, the shared CPUs are allocated. Shared CPUs can be overbooked, and container evictions occur when CPU resources are insufficient. Each node has an eviction threshold th_{evic}. When a node's remaining resources are insufficient, eviction is conducted based on the QoS level and eviction cost to free up resources. The above information related to the resources of tasks and nodes is available in the dataset, the details of which are in Section 5. Also, a discussion of how to obtain this data in a real system can be found in Section 6.

3.2 Modeling of Revenue, Cost, and Profit

To facilitate a more convenient overbooking evaluation, all costs are unified as the sum of the profit [20], [34].

Revenue: The revenue for all nodes can be calculated as follows:

$$\mathcal{I} = \sum_{n \in \mathbf{N}} \sum_{k \in \mathbf{K}} x_{k,n} \times v_k \times y_k, \tag{1}$$

where $x_{k,n} = 1$ indicates that task k is scheduled to node n. Otherwise, $x_{k,n} = 0$. Variable y_k represents whether task k was evicted, with 0 denoting no eviction and 1 denoting eviction. Eviction implies that the task is not completed, which naturally results in zero revenue [34].

Cost: The cost for all nodes can be calculated as follows:

$$\mathcal{C} = \sum_{n \in \mathbf{N}} \sum_{k \in \mathbf{K}} x_{k,n} \times \mathcal{C}_{k,n}, \tag{2}$$

where

$$\mathcal{C}_{k,n} = \mathcal{C}^b_n + \mathcal{C}^t_{k,n} + \mathcal{C}^c_{k,n} + \mathcal{C}^s_{k,n}. \tag{3}$$

In Eq. (3), $\mathcal{C}^b_n$ represents the node booting cost, which is proportional to the booting time of the node. Additionally, $\mathcal{C}^s_{k,n}$ denotes the container startup cost on node n. The transmission cost from task k to node n, $\mathcal{C}^t_{k,n}$, can be calculated as [30], [36], [37]:

$$\mathcal{C}^t_{k,n} = \omega_t \frac{z_k \left(|l^x_k - l^x_n|^2 + |l^y_k - l^y_n|^2\right)^{1/2}}{b_n}, \tag{4}$$

where $l_k = \{l^x_k, l^y_k\}$ and $l_n = \{l^x_n, l^y_n\}$ represent the locations of task k and node n, respectively. ω_t is a weight that controls the relationship between the cost and transmission time [36], [37]. b_n denotes the bandwidth of node n. Parameter $\mathcal{C}^c_{k,n}$ refers to the computation cost, which can be calculated based on the expected processing time and the extent of resource overbooking [20], [38]:

$$\mathcal{C}^c_{k,n} = \frac{\omega_c \times (d_k - t_k)}{\min\left(\frac{c_n - c^o_n(t)}{\sum_{p \in \{o_k = o^G | k \in \mathbf{P}_n(t)\}} c^r_k}, \frac{m_n - m^o_n(t)}{\sum_{k \in \{o_k = o^G | k \in \mathbf{P}_n(t)\}} m^r_k}, 1\right)}, \tag{5}$$

TABLE 1
Notations

Notation	Description
$\mathbf{N}$	Set of edge nodes
n	$n \in \mathbf{N}$ is an edge node
c_n	CPU capacity of edge node n
m_n	Memory capacity of edge node n
$c^u_n(t)$	CPU usage at time t
$m^u_n(t)$	Memory usage at time t
$\mathbf{P}_n(t)$	Running container list
l_n	location of edge node n
$\mathbf{P}$	Set of containers
p	$p \in \mathbf{P}$ is a container
c^r_p	CPU request of container p
m^r_p	Memory request of container p
$c^u_p(t)$	Real-time CPU usage
$m^u_p(t)$	Real-time memory usage
$\mathbf{K}$	Set of tasks
k	$k \in \mathbf{K}$ is a task
t_k	Arrival time of task k
d_k	Expected finish time of task k
z_k	Data size of task k
c^r_k	CPU request of task k
m^r_k	Memory request of task k
v_k	Value of task k
o_k	QoS level of task k
th_{evic}	Eviction threshold
$\mathcal{I}$	Revenue for all nodes
$x_{k,n}$	A variable indicating if task k is scheduled to node n
$\mathcal{C}$	Cost for all nodes
$\mathcal{C}^b_n$	Node booting cost
$\mathcal{C}^s_{k,n}$	Container startup cost on node n
$\mathcal{C}^t_{k,n}$	Transmission cost from task k to node n
$\mathcal{C}^c_{k,n}$	Computation cost of task k on node n
$c^o_n(t)$	Overbooked CPU resource
$m^o_n(t)$	Overbooked memory resource
$\mathcal{P}$	Total profit of the service provider
th	Overbooking threshold
$d_{k,n}$	Distance between task k and node n
s_t	State
a_t	Action
a^c_t	A discrete action for the container scheduling
a^r_t	A continuous action for the overbooking threshold
r	Reward function
$\mathcal{C}^{evic}_t$	Eviction cost of the evicted containers
π	Policy
$\mathcal{H}(\pi(\cdot\|s))$	Entropy to indicate the randomness of the policy π
$V(s_t)$	Value function
$L_\pi(\theta)$	Loss function

where $c^o_n(t)$ and $m^o_n(t)$ are the overbooked CPU and memory resources of node n, respectively. Parameter ω_c controls the weight of the cost and computation time. In Eq. (5), $c_n - c^o_n(t)$ denotes all CPU resources except those that are overbooked, and $\sum_{p \in \{o_k = o^G | k \in \mathbf{P}_n(t)\}} c^r_k$ denotes all CPU resources requested by tasks whose QoS level is Guaranteed. When $c_n - c^o_n(t) \geq \sum_{p \in \{o_k = o^G | k \in \mathbf{P}_n(t)\}} c^r_k$, it means that the resource overbooking does not affect the normal tasks. Otherwise, it means there are too many overbooked resources, which leads to an increase in the running time of the task. We can use $\frac{c_n - c^o_n(t)}{\sum_{p \in \{o_k = o^G | k \in \mathbf{P}_n(t)\}} c^r_k}$ to calculate the increase in task time due to overbooking [20].

Profit: According to the above definitions, the total profit of the service provider can be calculated as:

$$\mathcal{P} = \mathcal{I} - \mathcal{C}. \tag{6}$$

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

3.3 Problem Formulation

This subsection introduces the constraints, problem formulation, and problem analysis.

Constraints: To avoid the risk of excessive overbooking, a threshold th is set. If the threshold is exceeded, some containers must be evicted to release the resources such that the available resources are sufficient to deal with the burst requests. The overbooking constraint can be defined as:

$$c_n^u(t) + c_n^o(t) < th, m_n^u(t) + m_n^o(t) < th, \forall n \in \mathbf{N}. \quad (7)$$

In addition, when the total resource requests of the node reach a threshold, container eviction will also be performed to avoid risks, which can be denoted as:

$$c_n - \sum_{p \in \mathbf{P}_n(t)} c_p^r > th_{evic}^c, m_n - \sum_{p \in \mathbf{P}_n(t)} m_p^r > th_{evic}^m, \forall n \in \mathbf{N}, \quad (8)$$

where th_{evic}^c and th_{evic}^m are the container eviction thresholds.

Moreover, each task should be scheduled to only one edge node, which is represented as:

$$\sum_{n \in \mathbf{N}} x_{k,n} = 1, \forall k \in \mathbf{K}. \quad (9)$$

Problem Formulation: Our objective is to maximize the overall long-term profit, as defined in Eq. (6). The goal is to identify the optimal strategy that maximizes profit while adhering to the constraints. Consequently, the ROCS problem is formulated as follows:

Problem 1. $\max \mathcal{P} = \mathcal{I} - \mathcal{C}$,

$$\text{s.t.} \quad \textit{Eqs.}\ (7), (8), (9),$$
$$x_{k,n} \in \{0, 1\}, \forall n \in \mathbf{N}, \forall k \in \mathbf{K}.$$

Problem Analysis: Problem 1 is a sophisticated variant of the bin-packing problem, which is NP-hard and can be addressed using heuristic approaches. In this problem, the revenue $\mathcal{I}$ and the cost $\mathcal{C}$ can be expressed as the summation of $\mathcal{I}_t$ and $\mathcal{C}_t$ at each time t, respectively, i.e., $\mathcal{I} = \sum_t \mathcal{I}_t$, $\mathcal{C} = \sum_t \mathcal{C}_t$. Consequently, the profit $\mathcal{P}$ follows a first-order Markov process. Moreover, the first-order transition probability of resource demands remains quasi-static over extended periods and non-uniformly distributed, provided the time slice duration is appropriately chosen [39]. This observation makes RL algorithms suitable for this problem [40].

Among various RL algorithms, on-policy algorithms such as the Proximal Policy Optimization (PPO) algorithm [41] exhibit low sample efficiency. In contrast, off-policy algorithms such as Deep Deterministic Policy Gradient (DDPG) and extensions demonstrate higher sample efficiency [42]. However, they are sensitive to hyperparameters like learning rates and exploration constants. Furthermore, the integration of off-policy learning and neural networks poses significant challenges regarding stability and convergence in EC [43]. To address these issues, the soft actor-critic algorithm combines the stochastic policy advantages of PPO with the sampling efficiency of the DDPG algorithm [28]. This approach offers enhanced exploration capability and robustness in dynamic EC environments. Therefore, we propose an algorithm based on the soft actor-critic algorithm.

4 Algorithms

In this section, we first introduce the algorithm settings, followed by the proposal and analysis of the ROCS algorithm.

4.1 Algorithm Settings

The RL agent is responsible for making decisions. The state, action, reward, and policy must be defined to train an agent.

State: A state s_t provides a comprehensive description of the EC environment, primarily consisting of two components: the edge nodes and the arriving task. Firstly, the CPU usage $c_n^u(t)$, memory usage $m_n^u(t)$, and location of each edge node n are crucial, as they represent the resource usage and geographical distribution of edge nodes. Secondly, for each task, the requested CPU r_k^c and memory r_k^m play a significant role in decision-making. Additionally, the distance $d_{k,n} = \left(|l_k^x - l_n^x|^2 + |l_k^y - l_n^y|^2\right)^{1/2}$ between task k and edge node n substantially influences the task's completion time. Consequently, the state s_t can be defined as:

$$s_t = \{c_1^u(t), ..., c_{|\mathbf{N}|}^u(t), m_1^u(t), ..., m_{|\mathbf{N}|}^u(t), l_1, ..., l_{|\mathbf{N}|}, r_k^c, r_k^m, d_{k,n}\} \in \mathbf{S}, \quad (10)$$

where $\mathbf{S}$ is the set of all states.

Action: To optimize resource overbooking and container scheduling decisions from a global perspective, the action at each time t is defined as $a_t = \{a_t^c, a_t^r\} \in \mathbf{A}$, where $a_t^c \in \mathbf{N}$ represents a discrete action corresponding to the container scheduling decision, $a_t^r \in [0, 1]$ denotes a continuous action indicating the dynamic overbooking threshold associated with edge node a_t^c, and $\mathbf{A}$ refers to the action space.

Reward: The reward function r_t is of critical importance. The agent's objective is to maximize the reward, and our goal is to maximize profit; thus, the reward can be obtained as $r_t = \mathcal{P}_t$. Moreover, containers may be evicted. Consequently, the final reward at each time is modified to $r_t = \mathcal{P}_t - \mathcal{C}_t^{evic}$, where $\mathcal{C}_t^{evic}$ represents the eviction cost determined by the evicted containers.

Entropy: As introduced in Section 3.3, the proposed algorithm is based on the soft actor-critic to enhance the policy's robustness in adapting to the complex EC environment by incorporating entropy. Entropy is a measure of the randomness of a random variable. If X is a random variable and its probability density function is P, its entropy $\mathcal{H}$ is defined as [44]:

$$\mathcal{H}(X) = \mathbb{E}_{x \sim P}\left[-\log P(x)\right] \quad (11)$$

In RL, $\mathcal{H}(\pi(\cdot|s))$ can be employed to indicate the randomness of the policy π in state s. The concept is not only to maximize the cumulative reward but also to render the policy more random [44]. In this manner, an entropy regularization term is added to the RL objective:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_t r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right], \quad (12)$$

where α is a regularization coefficient that controls the significance of entropy. Entropy regularization increases the exploratory nature of the RL algorithm. A more significant α value enhances exploration, facilitating accelerated policy learning and reducing the likelihood of the policy becoming trapped in suboptimal local optima.

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

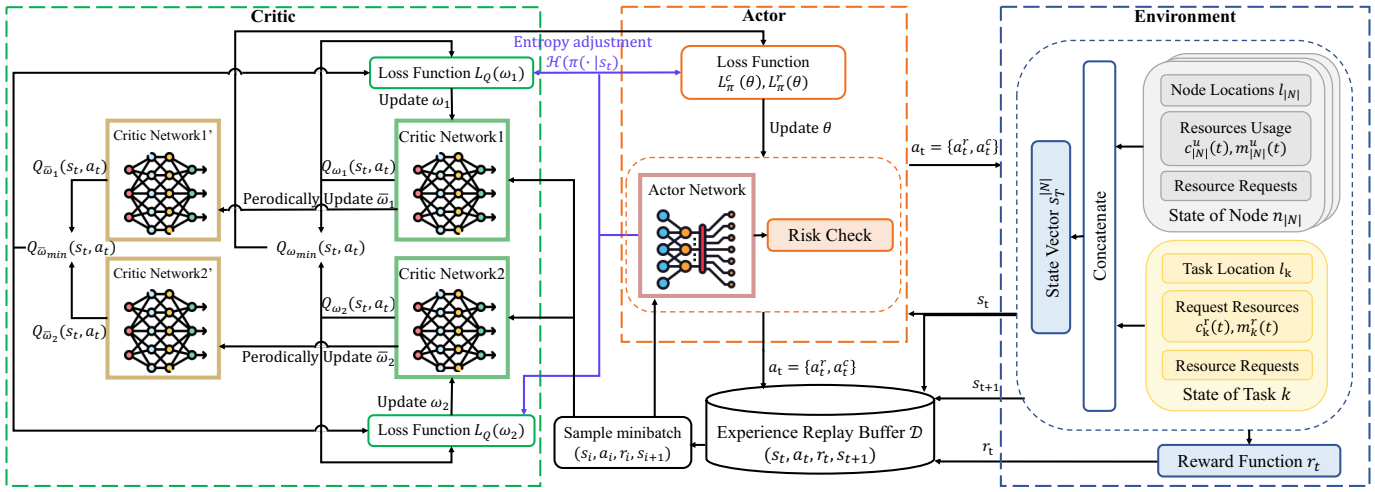


Fig. 2. Overview of ROCS algorithm

Policy: A policy is a rule used by the agent to decide what actions to take, typically denoted by π, i.e., $a_t \sim \pi(\cdot|s_t)$. To enhance the policy, the soft Bellman equation is required, defined as [28]:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}} \left[V(s_{t+1}) \right], \tag{13}$$

where γ represents the discount factor, and the value function $V(s_t)$ is calculated as follows:

$$\begin{aligned} V(s_t) &= \mathbb{E}_{a_t \sim \pi} \left[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \right] \\ &= \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t)] + \mathcal{H}(\pi(\cdot|s_t)). \end{aligned} \tag{14}$$

To jointly optimize resource overbooking and container scheduling decisions, the container scheduling decision a_t^c is assumed to primarily depend on the state s_t, while the overbooking threshold is more influenced by both the state s_t and the scheduling decision a_t^c, i.e., $\pi(a_t|s_t) = \pi(a_t^c|s_t)\pi(a_t^r|s_t, a_t^c)$. This is because container scheduling primarily relies on resource utilization, and the overbooking threshold of an edge node can only be determined appropriately after making the container scheduling decision, as introduced in Section 1. Therefore, $\mathcal{H}(\pi(\cdot|s_t))$ in Eq. (14) is modified to the joint entropy:

$$\begin{aligned} \mathcal{H}\left(\pi\left(a_t^c, a_t^r|s_t\right)\right) &= \alpha^c \mathcal{H}\left(\pi\left(a_t^c|s_t\right)\right) \\ &\quad + \alpha^r \sum_{a_t^c} \pi\left(a_t^c|s_t\right) \mathcal{H}\left(\pi\left(a_t^r|s_t, a_t^c\right)\right), \end{aligned} \tag{15}$$

where α^c and α^r are the weights. In EC, it is advantageous to assign different weights, as there would be a risk of one of these two entropies overshadowing the other, which could negatively impact exploration [27].

In accordance with Eq. (13), the soft policy evaluation can converge to the soft Q function of the policy π. The policy can be enhanced as follows.

$$\pi_{\text{new}} = \arg\min_{\pi'} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\cdot|s), \frac{\exp\left(\frac{1}{\alpha} Q^{\pi_{\text{old}}}(s, \cdot)\right)}{Z^{\pi_{\text{old}}}(s, \cdot)} \right), \tag{16}$$

where $\mathrm{D}_{\mathrm{KL}}(\cdot)$ denotes the Kullback-Leibler (KL) divergence. $Z^{\pi_{\text{old}}}(s, \cdot)$ normalizes the distribution. Generally, it does not contribute to the gradient and can be disregarded [28].

The soft policy iteration algorithm consists of two steps: soft policy evaluation based on Eqs. (13) and (16) and soft policy improvement, which has been shown to converge to an optimal maximum entropy policy regardless of the value of α [28]. The soft actor critic algorithm uses function approximators to approximate the Q-function and policy and optimizes the network parameters by stochastic gradient descent. So the soft actor critic algorithm also has good convergence [28]. In this paper, we also further prove the convergence of the algorithm in experiments.

4.2 ROCS Algorithm

Overview: The ROCS algorithm is presented in Algorithm 1. The input consists of the weights $\theta, \omega_1, \omega_2$ for the actor network and the two critic networks, respectively, while the output includes the policy π. As depicted in Lines 1 to 3, the weights $\bar{\omega}_1$ and $\bar{\omega}_2$ for the target networks and the replay memory $\mathcal{D}$ are initialized. The replay memory $\mathcal{D}$ stores training tuples. The initial state s_0 is then obtained, as shown in Line 5. As illustrated in Lines 6 to 14, for each time t, the action a_t is selected, and the overbooking risk is assessed by Algorithm 2. Actions of the first epoch are randomly selected to enhance the exploration of action selection. The reward r_t is computed, and the subsequent state s_{t+1} is observed. Then, as shown in Lines 15 to 22, network training is performed at the end of each epoch. The gradients are calculated, and the networks are updated. The details of the gradient step are provided in the following.

As depicted in Fig. 2, the ROCS algorithm employs two critic networks Q (with parameters ω_1 and ω_2) and an actor network π (with parameters θ) [28]. ROCS utilizes two Q networks, e.g., $Q_{\omega_1}(s_t, a_t)$ and $Q_{\bar{\omega}_1}(s_t, a_t)$, for each critic network but selects the one Q network with a smaller Q value at each time [45], thus mitigating the issue of Q value overestimation. The container scheduling decision a^c and the overbooking threshold a^r are determined according to the actor network. These decisions are then input to Algorithm 2 to verify and reduce the overbooking risk. Finally, the scheduling decisions are made, and the EC environment is updated accordingly.

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

Algorithm 1: ROCS

Input : $\omega_1, \omega_2, \theta$
Output: π
1 Initialize $Q_{\omega_1}(s,a)$, $Q_{\omega_2}(s,a)$, and $\pi_\theta(s)$;
2 Copy weights $\bar{\omega}_1 \leftarrow \omega_1$, $\bar{\omega}_2 \leftarrow \omega_2$;
3 Initialize replay memory $\mathcal{D} \leftarrow \emptyset$;
4 **for** *epoch* $\leftarrow$ *1, 2, ...* **do**
5 Get initial state s_0;
6 **for** $t = 1 \rightarrow T$ **do**
7 **if** $epoch = 1$ **then**
8 Randomly select the action $a_t = \{a_t^r, a_t^c\}$;
9 **else**
10 Select the action $a_t = \{a_t^r, a_t^c\} \sim \pi_\theta(s_t)$;
11 Call Algorithm 2 for risk check;
12 Calculate the reward r_t;
13 Get the next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$;
14 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$;
 `// Network Training`
15 **for** *each gradient step* **do**
16 Sample $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1,\dots,N}$;
17 Calculate y_i by Eq. (21);
18 Update ω_1 and ω_2 by Eq. (22);
19 Update θ by Eq. (18);
20 Update α^r and α^c by Eq. (20);
21 Update target networks by Eq. (23);
22 **end**

Loss: Losses are required to train the networks. The loss function of any Q function can be defined as [27], [28]:

$$\begin{aligned} &L_Q(\omega) \\ &=\mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\omega(s_t,a_t)-(r_t+\gamma V_{\bar{\omega}}(s_{t+1}))\right)^2\right] \\ &=\mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D},a_{t+1}\sim\pi_\theta(\cdot|s_{t+1})}\Bigg[\frac{1}{2}\Bigg(Q_\omega(s_t,a_t) \\ &\quad-\Bigg(r_t+\gamma\Bigg(\min_{j=1,2}Q_{\bar{\omega}_j}(s_{t+1},a_{t+1}) \\ &\quad-\sum_{a_{t+1}^c}\pi\left(a_{t+1}^c|s_{t+1}\right)\left(\alpha^c\log\pi\left(a_{t+1}^c|s_{t+1}\right)\right. \\ &\quad\left.+\alpha^r\pi\left(a_{t+1}^c|s_{t+1}\right)\log\pi\left(a_{t+1}^r|s_{t+1},a_{t+1}^c\right)\right)\Bigg)\Bigg)\Bigg)^2\Bigg] \end{aligned} \tag{17}$$

where $\mathcal{D}$ denotes the replay memory [46]. The target network $Q_{\bar{\omega}}$ is utilized to enhance the training stability [45].

For the continuous action, specifically the resource overbooking decision, traditional soft actor-critic algorithms output the mean and standard deviation of the Gaussian distribution. However, it is not differentiable to sample actions according to the Gaussian distribution. To solve this issue, the Beta distribution with shape parameters α and β is employed [47]. The Beta distribution has finite support and avoids the boundary effects associated with the Gaussian distribution. Consequently, it is bias-free and converges faster, leading to a quicker training process and higher performance. Furthermore, it is compatible with on-policy and off-policy algorithms such as soft actor-critic RL. The policy's loss function can be reformulated as follows:

$$\begin{aligned} L_\pi(\theta) =&\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi_\theta}\Big[\alpha\log\left(\pi_\theta\left(f_\theta\left(a_t;\alpha_\theta(s_t),\beta_\theta(s_t)\right)|s_t\right)\right) \\ &-\min_{j=1,2}Q_{\omega_j}\left(s_t,f_\theta\left(a_t;\alpha_\theta(s_t),\beta_\theta(s_t)\right)\right)\Big], \end{aligned} \tag{18}$$

where $f_\theta(\cdot)$ is a neural network transformation. With PyTorch, the reparameterization trick for the Beta distribution can be used to enable gradient computation [48].

Automating entropy adjustment: Different weights of entropy are required in various states. In a state where the optimal action is uncertain, the entropy value should be larger; otherwise, it should be smaller. To automatically adjust the entropy, the RL objective is altered as a constrained optimization problem, i.e., maximizing the expected return while constraining the mean of entropy to be greater than $\mathcal{H}_0$.

$$\begin{aligned} &\max_\pi \mathbb{E}_\pi\left[\sum_t r(s_t,a_t)\right] \\ &\text{s.t.}\quad \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[-\log\left(\pi_t(a_t|s_t)\right)\right]\geq\mathcal{H}_0 \end{aligned} \tag{19}$$

In EC, the parameter $\mathcal{H}_0$ is divided into $\mathcal{H}_0^c$ and $\mathcal{H}_0^r$ for container scheduling and resource overbooking decisions, respectively. To adjust the entropy automatically, the losses of α^c and α^r can be computed as follows.

$$\begin{aligned} L^c(\alpha) =&\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi(\cdot|s_t)}\sum_{a_t^c}\pi(a_t^c|s_t)\left[-\alpha^c\mathcal{H}_0^c\right. \\ &\left.-\alpha^c\log\pi\left(a_t^c|s_t\right)\right], \\ L^r(\alpha) =&\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi(\cdot|s_t)}\sum_{a_t^c}\pi(a_t^c|s_t)\left[-\alpha^r\mathcal{H}_0^r\right. \\ &\left.-\alpha^r\pi(a_t^c|s_t)\log\pi\left(a_t^r|s_t,a_t^c\right)\right]. \end{aligned} \tag{20}$$

Network training: With the loss and automating entropy adjustment, the networks can be trained. As shown in Lines 12 to 20 in Algorithm 1, several training tuples are first sampled for each gradient step. Then, y_i is computed as:

$$\begin{aligned} y_i =&r_i+\gamma\min_{j=1,2}Q_{\bar{\omega}_i}(s_{i+1},a_{i+1}) \\ &-\sum_{a_{i+1}^c}\pi_\theta(a_{i+1}^c|s_{i+1})\left(\alpha^c\log\pi_\theta\left(a_{i+1}^c|s_{i+1}\right)\right. \\ &\left.+\alpha^r\pi_\theta\left(a_{i+1}^c|s_{i+1}\right)\log\pi_\theta\left(a_{i+1}^r|s_{i+1},a_{i+1}^c\right)\right), \end{aligned} \tag{21}$$

where $a_{i+1} \sim \pi_\theta(\cdot|s_{i+1})$. And ω_1 and ω_2 are updated as:

$$L=\frac{1}{N}\sum_{i=1}^{N}\left(y_i-Q_{\omega_j}(s_i,a_i)\right)^2. \tag{22}$$

Finally, the target networks are also updated as follows.

$$\begin{aligned} \bar{\omega}_1 &\leftarrow \tau\omega_1+(1-\tau)\bar{\omega}_1, \\ \bar{\omega}_2 &\leftarrow \tau\omega_2+(1-\tau)\bar{\omega}_2, \end{aligned} \tag{23}$$

where τ is the weight that controls the update rate.

4.3 Risk Check

A risk check algorithm is proposed to address potential burst resource request situations to mitigate the overbooking risk further. As shown in Algorithm 2, the risk check

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

Algorithm 2: Risk Check

Input : $k, a_t^c, a_t^r, \mathbf{N}$
Output: a_t^c, r^a, r^c
1 $i \leftarrow 1, n \leftarrow a_t^c, r^a \leftarrow 0, r^c \leftarrow 0, j \leftarrow 1, \mathbf{N}_t \leftarrow \mathbf{N}$;
// Action Check
2 Sort $\mathbf{N}_t$ in descending order according to the action probability;
3 **while** $c_n^u(t) + r_k^c > a_t^r c_n$ **do**
4 Select the next node n from $\mathbf{N}_t$ and the corresponding threshold a_t^r;
5 $i \leftarrow i + 1$;
6 **if** $i > |\mathbf{N}|$ **then**
7 $n \leftarrow a_t^c, r^a \leftarrow 1$;
8 break;
// Container Eviction
9 Sort $\mathbf{P}_n(t)$ in descending order according to $\mathcal{C}_p^{evic}(t)$ and o_p as $\mathbf{P}'_n(t) = \{p'_1, ..., p'_{|\mathbf{P}|}\}$;
10 **while** $r_k^c + c_n^u(t) > th_{evic}^c c_n$ *or* $r_k^m + m_n^u(t) > th_{evic}^m m_n$ **do**
11 Evict the container p'_j;
12 $r^c \leftarrow r^c + r'_j, j \leftarrow j + 1$;
13 Update the available resources of the edge node;
14 $a_t^c \leftarrow n$;
15 **end**

algorithm primarily includes action check and container eviction.

Action check: First, the plausibility of the action is assessed, as illustrated in Lines 2 to 7 in Algorithm 2. Then, the feasibility of scheduling the container on this edge node is evaluated for the selected action a_t^c, i.e., the edge node n. If it is not feasible, another edge node n'_i is selected based on the action's probability. If all nodes are infeasible, the initially selected node is retained, and a significant negative reward is returned, i.e., $r^a \leftarrow 1$ indicates a substantial negative reward.

Container eviction: Subsequently, to ensure the resource availability of each edge node, the remaining resources based on the resource requests on the node are periodically inspected. As illustrated in Lines 8 to 12, the running containers are sorted according to the eviction cost $\mathcal{C}_p^{evic}(t) = \mathcal{T}_p \times (0.6 \times c_p^u(t) + 0.4 \times m_p^u(t))$ [49] and QoS level o_p, where $\mathcal{T}_p$ represents the container's running time. In other words, if the remaining requestable resources fall below the threshold, containers with QoS level o^B are evicted first, based on the eviction cost. If resources remain insufficient, containers with QoS level o^G are then evicted. Additionally, a negative reward r'_j is added, and the available resources for the edge node are updated.

4.4 Computational Complexity Analysis

The computational complexity analysis is as follows. The primary process of the ROCS algorithm consists of Algorithm 1 and Algorithm 2. First, as depicted in Algorithm 1, the state is obtained by Eq. (10), and the complexity is $O(|\mathbf{N}|)$. Then, the action is selected according to the policy π_θ. The time complexity of the networks is only related to the network size, which can be considered a constant time O_t. Thus, the complexity of action selection is $O(O_t)$. The training complexity can be expressed in floating-point operations (FLOPs) [50]. The training complexity does not affect the decision-making complexity of the network, so it can be considered polynomial time. Next, as depicted in Algorithm 2, the time complexities of the action check and container eviction are $O(|\mathbf{N}|)$ and $O(|\mathbf{P}_n(t)|)$, respectively. These steps are executed sequentially, allowing them to be completed in polynomial time.

Moreover, to evaluate the complexity of the network training, a theoretical analysis of the computational complexity of the neural networks based on FLOPs is performed, which is widely used to measure the computational complexity of deep learning models [4]. As shown in Fig. 2, the ROCS algorithm contains four critic networks with the same structure and one actor network. The actor network contains three hidden fully-connected layers and two output layers, for five fully-connected layers. Assuming that the input and output dimensions of the j-th layer are H_i^j and H_o^j, respectively. Then the total FLOPs for the actor network are $\sum_{j=1}^{7}(2H_i^j - 1)H_o^j$ [51]. Each critic network contains two fully-connected layers, so the FLOPs for each critic network are $\sum_{j=1}^{2}(2H_i^j - 1)H_o^j$. Usually, a linear layer is followed by a non-linear activation function, such as a ReLU or a Softmax [52]. It is common not to count these operations, as they only take up a tiny fraction of the overall time. For example, a ReLU is just $y = \max(x, 0)$. On a fully connected linear layer with H_o^j output neurons, the ReLU uses H_o^j of these computations, i.e., it has H_o^j FLOPs. Compared with matrix multiplies and inner products, the FLOPs of the activation function can be ignored.

To summarize, our algorithm yields results in polynomial time. Moreover, our experiments also demonstrate that the execution time of the ROCS algorithm is acceptable, as shown in Section 5.2.

5 Evaluation

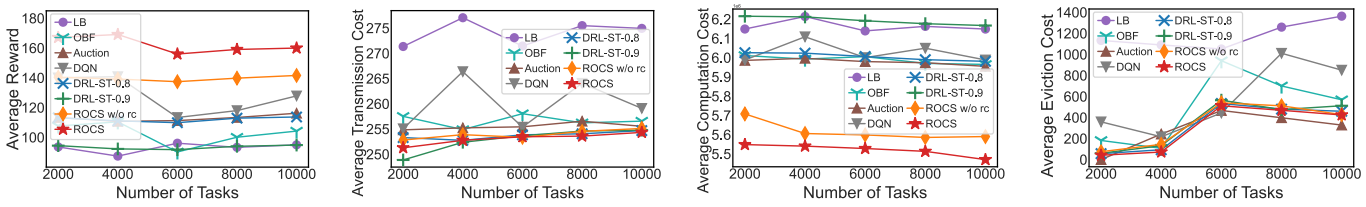
In this section, the performance of the proposed algorithm is evaluated. First, the experimental settings are introduced, followed by the presentation and analysis of the results.

5.1 Experimental Settings

Data preprocessing: The real-world trace used in the experiments is collected from a large production cluster on approximately 1800 machines in Alibaba's artificial intelligence platform, spanning July and August 2020 [8]. According to the dataset, the edge node CPU capacities are set to 64 or 96 cores, and memory capacities are set to 512 or 1024 GB. Node locations are randomly generated following a uniform distribution, with coordinates in the range $[0, 1]$ unit. To generate tasks, specific task features are extracted from the dataset, including start time, end time, requested CPU, requested memory, real-time CPU usage, real-time memory usage, and task data size [53]. The expected processing time of a task is obtained by subtracting the start time from the end time. Task locations are also randomly generated within the $[0, 1]$ range, and QoS levels are randomly assigned. Tasks arrive randomly, and the distance between tasks and nodes is calculated using the Euclidean distance formula.

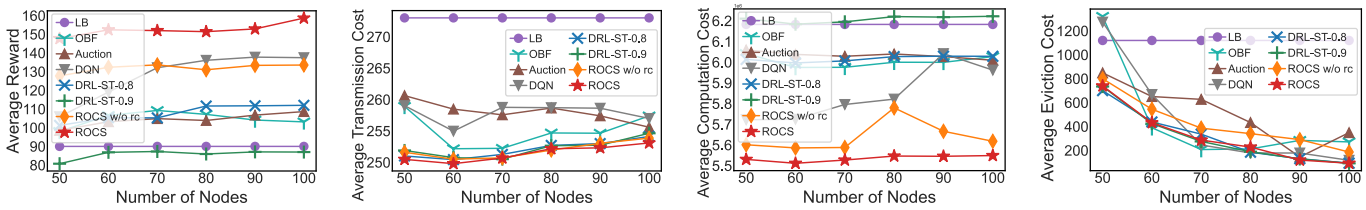
Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936



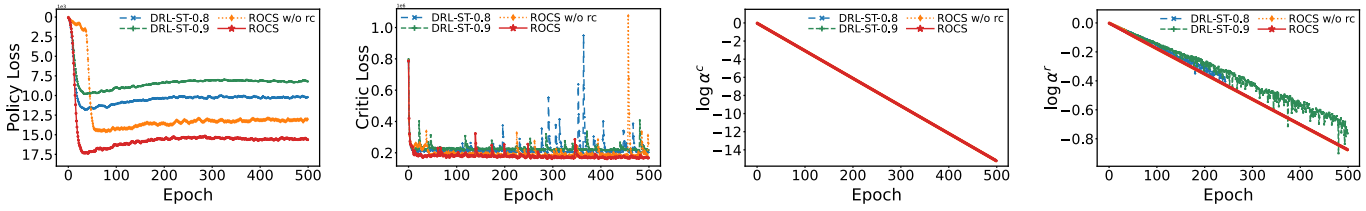
(a) Reward (b) Latency Cost (c) Computation Cost (d) Eviction Cost

Fig. 3. Performance with different number of tasks



(a) Reward (b) Latency Cost (c) Computation Cost (d) Eviction Cost

Fig. 4. Performance with different number of nodes



(a) Policy Loss (b) Critic Loss (c) $\log \alpha^c$ (d) $\log \alpha^r$

Fig. 5. Losses of the ROCS algorithm

Baselines: To compare performance, several baselines are employed. The action spaces of these algorithms are identical to that of the ROCS algorithm. The details are as follows.

1) **LB**: Load Balancing (LB) is a load-balancing algorithm that selects a node with the most available resources [50].
2) **OBF**: Overbooking Factor (OBF) is a traditional overbooking algorithm [11]. Each time, a node is selected according to an overbooking factor defined as $OBF = \min\left(\frac{\sum_{p\in\mathbf{P}_n(t)} c_p^r - c_n^u(t)}{\min\left(c_n, \sum_{p\in\mathbf{P}_n(t)} c_p^r\right)}, \frac{\sum_{p\in\mathbf{P}_n(t)} m_p^r - m_n^u(t)}{\min\left(m_n, \sum_{p\in\mathbf{P}_n(t)} m_p^r\right)}\right)$.
3) **Auction** [20]: It is an auction-based pricing algorithm to determine the overbooking threshold.
4) **DQN** [26]: Deep Q-Network (DQN) is a value-based RL algorithm.
5) **DRL-ST-0.9**: Deep Reinforcement Learning with Static Threshold $th = 0.9$ (DRL-ST-0.9) is a DRL algorithm based on soft actor-critic [28]. The overbooking threshold is static and set to 0.9.
6) **DRL-ST-0.8**: Deep Reinforcement Learning with Static Threshold $th = 0.8$ (DRL-ST-0.8) [28].
7) **ROCS w/o rc**: ROCS algorithm without risk check (ROCS w/o rc) is an algorithm without Algorithm 2 risk check based on the ROCS algorithm.

Parameter settings: The target entropy $\mathcal{H}_0^r$ and $\mathcal{H}_0^c$ are set to -0.25 and 0.25, respectively. The weights α^r and α^c are automatically adjusted during training. The learning rates for the actor and critic networks are set to 1×10^{-4}. The discount factor γ is set to 0.9, and the target network update parameter τ is set to 0.1. CPU and memory resource capacities are scaled into $(0, 1]$ using the min-max normalization method. The batch size is set to 256. Three fully-connected layers are used for the actor network, with dimensions of 1024 and 512. A beta distribution is employed to describe the probability density distribution of the overbooking threshold, with its two parameters, α and β, represented by 512-dimensional fully connected networks. Each critic network consists of two fully connected layers with a dimension of 1024. The actor network is updated at the end of each epoch. Following the approach in [34], v_k is set to a value inversely related to computation time, e.g., $900 - c_{k,n}^c$. The eviction cost is set as $\mathcal{C}_t^{evic} = \sum_{p\in\mathbf{P}^{evic}(t)} \mathcal{C}_p^{evic}(t)$, where $\mathbf{P}^{evic}(t)$ is the set of evicted containers at time t. Additionally, to minimize the number of evictions, an additional penalty term $\mathcal{C}_t^{evic} \times (1 + |\mathbf{P}^{evic}(t)|)$ is added.

Simulator setup: The EC simulation environment is im-

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

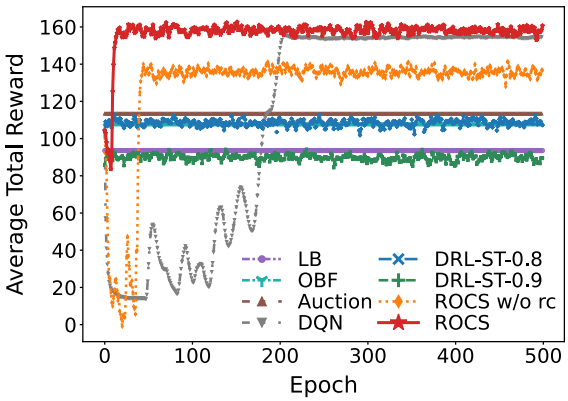


Fig. 6. Reward

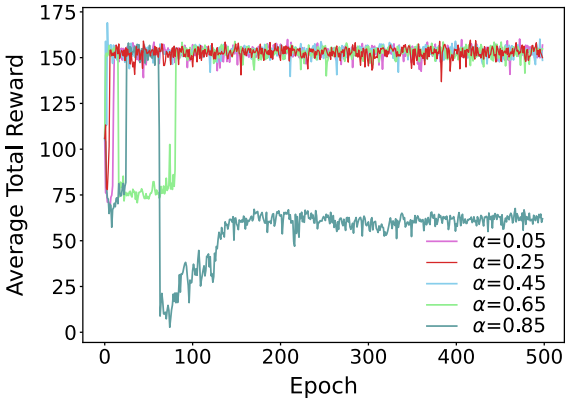
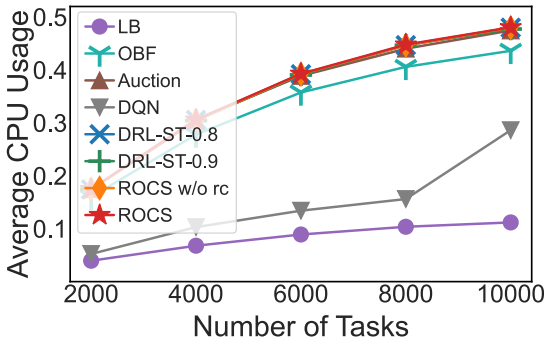
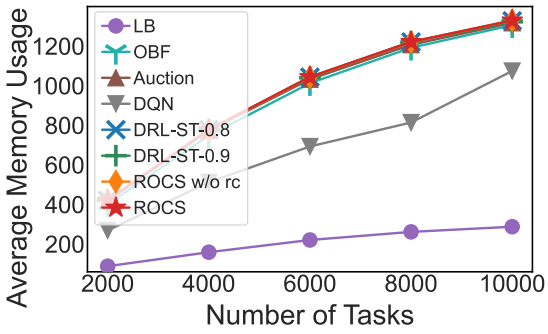


Fig. 7. Reward with different α



(a) CPU Usage



(b) Memory Usage

Fig. 8. CPU and memory usage with different task numbers

plemented with Python, which mainly includes the classes of edge node, task, scheduler, etc. The main classes are as follows.

1) Node. Each node has a node ID. The features of a node include the CPU capacity, memory capacity, location, bandwidth, available CPU, available memory, and a list of running tasks.
2) Task. The task class includes the task ID, data size, CPU request, memory request, start time, end time, location, QoS level, etc.

The EC environment is created based on these classes to return the reward, state, etc. Moreover, the environment is updated online according to the action the agent selects. The simulation is run in NVIDIA DGX A100 with AMD EPYC 7742 CPU. And the operating system is CentOS 7.

5.2 Experimental Results

Multiple sets of experiments are conducted and analyzed to demonstrate the ROCS algorithm's effectiveness.

Performance with the different number of tasks: The reward, latency cost, computation cost, and container eviction cost with varying task numbers are presented in Fig. 3. As depicted in Fig. 3(a), the ROCS algorithm consistently outperforms other baselines under different numbers of tasks. Moreover, the ROCS algorithm has increased profit by 75%, 59%, 44%, 27%, 74%, 45%, and 16% compared with the LB, OBF, Auction, DQN, DRL-ST-0.9, DRL-ST-0.8, and ROCS w/o rc algorithms, respectively.

As shown in Fig. 3(b), regarding latency, the ROCS algorithm is comparable to the algorithms with static thresholds and outperforms the LB and OBF algorithms. The computation cost is shown in Fig. 3(c). The ROCS algorithm significantly reduces computation costs since the state contains the resource status of each node. As illustrated in Fig. 3(d), since the ROCS algorithm automatically adjusts the overbooking threshold, the eviction cost is significantly lower than other baselines, especially for the DRL algorithms with static thresholds. This highlights the advantages of the ROCS algorithm's hybrid action space and the risk check mechanism, which can fully consider the impact of overbooking and container scheduling while effectively reducing the eviction number.

Performance with the different number of nodes: As illustrated in Fig. 4(a), the rewards of different algorithms exhibit an overall increasing trend as the number of edge nodes increases. This is because, with more edge nodes, the total computing resources increase, leading to a decrease in container evictions, as shown in Fig. 4(d). Furthermore, as depicted in Figs. 4(b) and 4(c), the ROCS algorithm's latency cost and computation cost remain superior to the LB and OBF algorithms. Overall, when the number of nodes changes, the reward of the ROCS algorithm is, on average, 69%, 46%, 78%, 41%, and 16% higher than the LB, OBF, DRL-ST-0.9, DRL-ST-0.8, and ROCS w/o rc baseline algorithms, respectively.

Performance of the training process: Fig. 5 displays

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

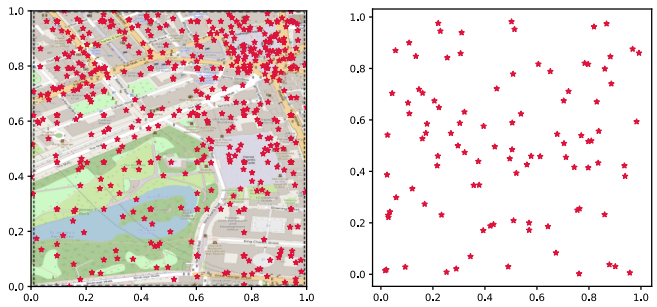
TABLE 2
Execution resources and execution time for different algorithms

Algorithm	RAM	VRAM	Execution Time (s)
LB	-	-	$4.23 \times 10^{-}5$
OBF	-	-	$5.27 \times 10^{-}5$
Auction	-	-	$1.81 \times 10^{-}4$
DQN	155Kb	90Kb	$4.51 \times 10^{-}4$
DRL-ST-0.8	67Kb	109Kb	$1.28 \times 10^{-}3$
DRL-ST-0.9	67Kb	109Kb	$1.29 \times 10^{-}3$
ROCS w/o rc	59Kb	70Kb	$1.31 \times 10^{-}3$
ROCS	59Kb	70Kb	$1.34 \times 10^{-}3$

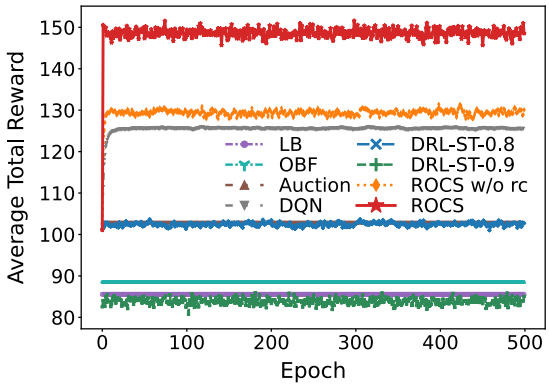
the different losses of ROCS, DRL-ST-0.9, DRL-ST-0.8, and ROCS w/o rc algorithms. It can be seen from Figs. 5(a) and 5(b) that these algorithms have reached convergence after training for approximately 100 epochs. This illustrates the good convergence performance of the ROCS algorithm. In addition, the change of α also determines the performance of the algorithm. A larger α generally leads to a more random strategy and vice versa. Figs. 5(c) and 5(d) show $\log \alpha^c$ and $\log \alpha^r$, respectively. As the number of training epochs increases, $\log \alpha^c$ and $\log \alpha^r$ decrease, consistent with common sense. Because as the training process gradually converges, the randomness of the policy decreases, and the action selection tends to choose better actions. This further illustrates the effectiveness of our algorithm. Moreover, the rewards of different algorithms are shown in Fig. 6. It can be seen that the ROCS algorithm has reached convergence when the training is fewer than 100 epochs, and the performance of the subsequent epochs is stable. Overall, the ranking of algorithm performance is ROCS > DQN > ROCS w/o rc > Auction > DRL-ST-0.8 > OBF > LB > DRL-ST-0.9.

Moreover, to further verify the convergence of the proposed ROCS algorithm, we conduct comparison experiments with different initial values for α. As can be seen in Fig. 7, when α takes a smaller value, the algorithm is more effective and converges faster. When α becomes larger, such as $\alpha = 0.85$, the reward is smaller, but the algorithm still reaches convergence in about 200 epochs. In our ROCS algorithm, we have taken $\alpha = 0.25$, which is a value that allows the algorithm to converge faster and perform better. In short, Fig. 7 further shows that our ROCS algorithm has good convergence.

CPU and memory usage: Fig. 8 presents the resource usage of edge nodes. As the number of tasks increases, the CPU usage of the ROCS algorithm becomes higher. Since the LB algorithm does not consider overbooking and polls for each node, it has low CPU and memory utilization. The DQN algorithm does not consider the joint decision-making of overbooking thresholds and container scheduling well, and training and updating based on the maximal Q introduces a large error, so it has a relatively low performance. The OBF algorithm considers container scheduling based on overbooking but does not adjust thresholds dynamically well, so the CPU utilization is low. Compared with the LB and OBF algorithms, CPU usage is significantly improved by 339% and 10%, respectively. In Fig. 8(b), the memory usage of the ROCS algorithm is also maintained at a high level. Memory usage is improved by 373% and 3%, respectively, compared with the LB and OBF algorithms. For



(a) Real-world edge nodes (b) Random edge nodes



(c) Reward

Fig. 9. Locations and rewards with real-world physical distribution for edge nodes. (a) The area of the United Kingdom (0.9km $\times$ 0.9km bounded by the coordinate pairs $[-0.135, 51.500]$ and $[-0.127, 51.509]$) with 719 nodes (crimson starts) [54]. (b) Generated random edge nodes with uniform distribution.

other algorithms, the CPU and memory utilization are closer to the ROCS algorithm but perform worse in other metrics, e.g., the transmission, computation, and eviction costs. Therefore, overall, our proposed ROCS is the most effective algorithm in all aspects combined. This fully demonstrates the necessity of overbooking and the effective enhancement of resource usage through overbooking.

Computational complexity and execution resources: To further demonstrate the computational complexity of the algorithm, we have executed the algorithms 5000 times, and the results of the average decision-making time are presented in TABLE 2. As observed from TABLE 2, the algorithms with the least execution time are LB and OBF due to their lack of complex neural network reasoning processes. The ROCS algorithm requires 1.34×10^{-3} seconds for a single decision within acceptable limits. Moreover, compared with the DRL-ST-0.8 and DRL-ST-0.9 algorithms, the incremental decision-making time incurred by the neural network addition in the ROCS algorithm is virtually negligible. Hence, we can conclude that the computational complexity of the ROCS algorithm is acceptable.

Moreover, we record the random access memory (RAM) and video RAM (VRAM) required for the algorithms to run in TABLE 2. The LB, OBF, and Auction algorithms require minimal computation resources and are, therefore, not recorded. We used the tool $torch.profiler$ for documentation [55] for other algorithms. As shown in TABLE

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

2, the DQN algorithm requires the most RAM for decision-making, while our proposed ROCS algorithm requires the least RAM and VRAM. Therefore, the ROCS algorithm requires significantly less computation resources and can run effectively in EC.

Performance with real-world distribution for edge nodes. To further illustrate the performance of the proposed ROCS algorithm, we use real-world edge node distribution data located in the UK [54], as shown in Fig. 9(a). As a comparison, Fig. 9(b) shows our randomly generated edge nodes. Fig. 9(c) shows the experimental results based on the distribution of real-world edge nodes. As can be seen from Fig. 9(c), our ROCS algorithm achieves good results even with real-world edge node distribution. Overall, the order of effectiveness of the algorithms is ROCS $>$ ROCS w/o rc $>$ LB $>$ others.

6 Discussions

From the experimental results, we can see the effectiveness of our algorithms. However, the following issues deserve further investigation.

Potential issues in EC: This paper introduces a joint resource overbooking and container scheduling algorithm based on soft actor-critic RL. This innovative approach enhances the resource utilization efficiency of edge nodes effectively. However, the algorithm's design fundamentally relies on centralized decision-making, which inadequately addresses the network instability in EC, e.g., network connection interruptions and recovery. This issue could be mitigated by employing multi-agent RL, i.e., deploying a decision-making agent at each edge node [56]. Furthermore, data management and security are other significant considerations in EC [57], [58], although they do not constitute the focus of this paper.

Moreover, how to collect some necessary data in the actual EC system is also an important issue. First, the data size of the task can be obtained either before the transmission or after the transmission is completed. After obtaining the data size and the requested resource information such as CPU and memory, a prediction of the expected processing time of the current task can be made based on the historical task information [4]. In addition, since we focus on the approach of centralized decision, the location information (e.g., latitude and longitude) of each node can be transmitted to the central node [36]. Generally, the further the distance between two edge nodes, the corresponding network hop count and delay are larger, so the delay between nodes can be estimated from the distance [30].

Threshold-based overbooking method: The threshold-based overbooking method offers several advantages: 1) Simplicity and ease of implementation: It relies on setting specific usage thresholds, e.g., CPU utilization, making the management and monitoring processes more intuitive. 2) Flexibility: Thresholds can be adjusted based on actual needs, providing administrators with flexible resource management capabilities. 3) Responsiveness to changes in resource demand: When resource usage reaches or exceeds the set thresholds, the system can quickly respond by adding more resources or migrating workloads. 4) Cost-effectiveness: Proper setting and management of thresholds can maximize resource utilization while minimizing waste. 5) Reduced resource competition and conflicts: Appropriate threshold settings can help avoid excessive competition for limited resources, reducing the risk of performance bottlenecks and service interruptions. 6) Adaptability: This method is versatile and can be applied in various systems. Therefore, many current studies are based on threshold-based methods, as described in Section 2.

However, the threshold-based overbooking method also has limitations, such as being unable to cope with sudden changes in resource demands and a heavy reliance on the accuracy of threshold settings. Therefore, in this paper, we combine the threshold-based approach with container scheduling and dynamically adjust the thresholds to realize flexible and efficient resource management. Future work will consider combining the threshold-based approach with other approaches, such as prediction and prioritization, to achieve more efficient resource management.

Implementation method: The ROCS algorithm can be implemented on Kubernetes through the scheduling framework [59]. During deployment, Prometheus collects and stores the real-time resource load data of the edge cluster, subsequently accessed via the Kubernetes API [60]. The Kubernetes scheduling framework allows for complete scheduling cycle customization, entailing two steps: the Scheduling Cycle and the Binding Cycle. The former selects an edge node for the container, while the latter applies that decision to the edge cluster. These two cycles collectively form a "scheduling context" featuring multiple extension points, including PreFilter, Filter, PostFilter, PreScore, Score, NormalizeScore, Reserve, Permit, PreBind, Bind, PostBind, etc. To enact a custom scheduling algorithm, a scheduler plugin must be implemented, with several extension points registered [61]. Once the scheduler plugin is completed, it can be deployed as a deployment through Kubernetes, with the application scheduler specified as our custom scheduler [62].

Nevertheless, this comprehensive deployment process necessitates a substantial amount of engineering code. Additionally, using RL demands interaction between the training and decision-making processes, implicating RL and the Kubernetes API, which makes system implementation more complex. As this paper primarily focuses on the scheduling algorithm, the effect of the algorithm has been validated through large-scale simulations rather than implementation in Kubernetes. Currently, our team is working on deploying a custom scheduler into the Kubernetes system. This endeavor has surfaced many new challenges, guiding our future work.

7 Conclusion

We have considered the resource constraints, resource heterogeneity, and geographical distribution of edge nodes and formulated the ROCS problem to investigate overbooking in EC networks. An efficient algorithm based on soft actor-critic reinforcement learning was proposed to explore a hybrid action space for solving the ROCS problem. We also introduced a risk check mechanism to mitigate potential overbooking risks. The experimental results highlighted the performance advantages of our ROCS algorithm in

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

enhancing resource utilization in EC networks compared to the existing schemes. In future work, we will examine overbooking in cloud-edge collaboration scenarios, where the cloud can effectively supplement the computing power of edge nodes. Furthermore, we will consider the overbooking of dependent tasks and investigate the impact of constrained bandwidth in EC.

ACKNOWLEDGMENTS

This work is supported in part by the Chinese National Research Fund (NSFC) under Grant 62302048 and 62272050; in part by FDCT 0158/2022/A; in part by Open Foundation of Yunnan Key Lab of Software Engineering under Grant 2023SE207; in part by Guangdong Key Lab of AI and Multi-modal Data Processing, UIC under Grant 2020KSYS007; and in part by Interdisciplinary Intelligence SuperComputer Center of Beijing Normal University (Zhuhai).

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] S. Hu, W. Shi, and G. Li, "Cec: A containerized edge computing framework for dynamic resource provisioning," *IEEE Transactions on Mobile Computing*, 2022, doi: 10.1109/TMC.2022.3147800.

[3] L. Pan, L. Wang, S. Chen, and F. Liu, "Retention-aware container caching for serverless edge computing," in *Proceedings of the 41st IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 1069–1078.

[4] Z. Tang, J. Lou, and W. Jia, "Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3444–3459, 2023.

[5] Resource management for pods and containers. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the third ACM Symposium on Cloud Computing (SoCC)*, 2012, pp. 1–13.

[7] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "Rose: Cluster resource scheduling via speculative over-subscription," in *Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 949–960.

[8] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 945–960.

[9] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in a shared internet hosting platform," *ACM Transactions on Internet Technology*, vol. 9, no. 1, pp. 1–45, 2009.

[10] The linux kernel supports the following overcommit handling modes. [Online]. Available: https://www.kernel.org/doc/Documentation/vm/overcommit-accounting

[11] L. Tomás and J. Tordsson, "Improving cloud infrastructure utilization through overbooking," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing conference (CAC)*, 2013, pp. 1–10.

[12] Overcommitting. [Online]. Available: https://docs.openshift.com/container-platform/3.11/admin_guide/overcommit.html

[13] Oversubscribe resources. [Online]. Available: https://issues.apache.org/jira/browse/MESOS-354

[14] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit: peak prediction-driven resource overcommitment in datacenters," in *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys)*, 2021, pp. 556–573.

[15] P. Rahimzadeh, Y. Im, G. Jung, C. Joe-Wong, and S. Ha, "Echo: efficiently overbooking applications to create a highly available cloud," in *Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 1–11.

[16] H. Yu, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Accelerating serverless computing by harvesting idle resources," in *Proceedings of the ACM Web Conference 2022 (WWW)*, 2022, pp. 1741–1751.

[17] H. Huang, Y. Zhao, J. Rao, S. Wu, H. Jin, D. Wang, K. Suo, and L. Pan, "Adapt burstable containers to variable cpu resources," *IEEE Transactions on Computers*, 2022, doi: 10.1109/TC.2022.3174480.

[18] M. Liwang and X. Wang, "Overbooking-empowered computing resource provisioning in cloud-aided mobile edge networks," *IEEE/ACM Transactions on Networking*, 2022, doi: 10.1109/TNET.2022.3167396.

[19] M. Liwang, R. Chen, X. Wang, and X. Shen, "Unifying futures and spot market: Overbooking-enabled resource trading in mobile edge networks," *IEEE Transactions on Wireless Communications*, 2022, doi: 10.1109/TWC.2022.3141094.

[20] Z. Tang, F. Zhang, X. Zhou, W. Jia, and W. Zhao, "Pricing model for dynamic resource overbooking in edge computing," *IEEE Transactions on Cloud Computing*, 2022, doi: 10.1109/TCC.2022.3175610.

[21] F. Zhang, Z. Tang, M. Chen, X. Zhou, and W. Jia, "A dynamic resource overbooking mechanism in fog computing," in *Proceedings of the 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2018, pp. 89–97.

[22] B. Sun, Y. Jiang, Y. Wu, Q. Ye, and D. H. Tsang, "Performance analysis of mobile cloud computing with bursty demand: A tandem queue model," *IEEE Transactions on Vehicular Technology*, 2022, doi: 10.1109/TVT.2022.3178634.

[23] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proceedings of the 37th IEEE Conference on Computer Communications (INFOCOM)*, 2018, pp. 468–476.

[24] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, "Layer aware microservice placement and request scheduling at the edge," in *Proceedings of the 40th IEEE Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–9.

[25] C. Xie, B. Yu, Z. Zeng, Y. Yang, and Q. Liu, "Multilayer internet-of-things middleware based on knowledge graph," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2635–2648, 2020.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] O. Delalleau, M. Peter, E. Alonso, and A. Logut, "Discrete and continuous action representation for practical rl in video games," *arXiv preprint arXiv:1912.11077*, 2019.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 1861–1870.

[29] S. Hu, Z. Qu, B. Tang, B. Ye, G. Li, and W. Shi, "Joint service request scheduling and container retention in serverless edge computing for vehicle-infrastructure collaboration," *IEEE Transactions on Mobile Computing*, 2023.

[30] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, "Multi-user layer-aware online container migration in edge-assisted vehicular networks," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2023, doi: 10.1109/TNET.2023.3330255.

[31] K. sigs. Trimaran: Load-aware scheduling plugins. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/pkg/trimaran

[32] L. Zanzi, J. X. Salvat, V. Sciancalepore, A. G. Saavedra, and X. Costa-Perez, "Overbooking network slices end-to-end: Implementation and demonstration," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018, pp. 144–146.

[33] J. Gao, B. Hu, J. Liu, H. Wang, Q. Huang, and Y. Zhao, "Overbooking-enabled task scheduling and resource allocation in mobile edge computing environments." *Intelligent Automation & Soft Computing*, vol. 37, no. 1, 2023.

[34] Y. Yang and H. Shen, "Deep reinforcement learning enhanced greedy optimization for online scheduling of batched tasks in cloud hpc systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3003–3014, 2021.

[35] Control cpu management policies on the node. [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/

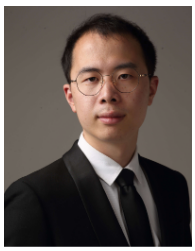
Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

[36] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2019.
[37] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.
[38] S. S. Mondal, N. Sheoran, and S. Mitra, "Scheduling of time-varying workloads using reinforcement learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, no. 10, 2021, pp. 9000–9008.
[39] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. Lau, "Dynamic virtual machine management via approximate markov decision process," in *Proceedings of the 35th IEEE International Conference on Computer Communications (INFOCOM)*, 2016, pp. 1–9.
[40] Y. Li, Y. Wu, Y. Song, L. Qian, and W. Jia, "Dynamic user-scheduling and power allocation for swipt aided federated learning: A deep learning approach," *IEEE Transactions on Mobile Computing*, 2022, doi: 10.1109/TMC.2022.3201622.
[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[42] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[43] H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Proceedings of the 22nd International Conference on Neural Information Processing Systems (NeurIPS)*, 2009, pp. 1204–1212.
[44] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 1433–1438.
[45] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 30, no. 1, 2016, pp. 2094–2100.
[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[47] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, pp. 834–843.
[48] Probability distributions - torch.distributions. [Online]. Available: https://pytorch.org/docs/stable/distributions.html
[49] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: a comprehensive review," *Journal of Grid Computing*, vol. 18, no. 1, pp. 1–42, 2020.
[50] Q. Liu, T. Xia, L. Cheng, M. Van Eijk, T. Ozcelebi, and Y. Mao, "Deep reinforcement learning for load-balancing aware network control in iot edge systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1491–1502, 2021.
[51] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2016.
[52] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
[53] Alibaba cluster trace gpu v2020. [Online]. Available: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-gpu-v2020
[54] Opencellid. [Online]. Available: https://opencellid.org/downloads.php?token=pk.3726ac9809881e1f346e97a8e842e9c0
[55] Pytorch documentation. [Online]. Available: https://pytorch.org/docs/
[56] Y. Zhang, B. Di, Z. Zheng, J. Lin, and L. Song, "Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning," *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2565–2578, 2020.
[57] Y. Li, W. Liang, W. Xu, Z. Xu, X. Jia, Y. Xu, and H. Kan, "Data collection maximization in iot-sensor networks via an energy-constrained uav," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 159–174, 2021.
[58] D. Nguyen, M. Ding, P. Pathirana, A. Seneviratne, J. Li, and V. Poor, "Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2021–2037, 2023.
[59] Scheduling framework. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/
[60] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.
[61] Scheduler plugins. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins
[62] Deployments. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Zhiqing Tang received the B.S. degree from School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and the Ph.D. degree from Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2022. He is currently an assistant professor with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, China. His current research interests include edge computing, resource scheduling, container scheduling, and reinforcement learning.

Fangyi Mou received the B.S. degree from Department of Electronic, Communication and Physics, Shandong University of Science and Technology, China, in 2017 and the M.Sc. degree from Department of Computer Science and Engineering, University of Macau, China, in 2020. She is currently working as a research assistant with Institute of Artificial Intelligence and Network, Beijing Normal University, China. Her current research interests include mobile edge computing, resource allocation, and reinforcement learning.



Jiong Lou received the B.S. degree and Ph.D. degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2016 and 2023. Since 2023, he has held the position of research assistant professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He has published more than ten papers in leading journals and conferences (e.g., ToN, TMC and TSC). His current research interests include edge computing, task scheduling and container management. He has served as a reviewer for CN, JPDC, IoT-J, and ICDCS.

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3386936

Weijia Jia (Fellow, IEEE) is currently a Chair Professor, Director of BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (Zhuhai) and VP for Research of BNU-HKBU United International College (UIC) and has been the Zhiyuan Chair Professor of Shanghai Jiao Tong University, China. He was the Chair Professor and the Deputy Director of State Kay Laboratory of Internet of Things for Smart City at the University of Macau. He received BSc/MSc from Center South University, China, in 82/84 and Master of Applied Sci./PhD from Polytechnic Faculty of Mons, Belgium in 92/93, respectively, all in computer science. From 93-95, he joined German National Research Center for Information Science (GMD) in Bonn (St. Augustine) as a research fellow. From 95-13, he worked at City University of Hong Kong as a professor. His contributions have been recognized as optimal network routing and deployment, anycast and QoS routing, sensors networking, AI (knowledge relation extractions; NLP, etc.), and edge computing. He has over 600 publications in the prestige international journals/conferences and research books, and book chapters. He has received the best product awards from the International Science & Tech. Expo (Shenzhen) in 2011/2012 and the 1st Prize of Scientific Research Awards from the Ministry of Education of China in 2017 (list 2). He has served as area editor for various prestige international journals, chair and PC member/skeynote speaker for many top international conferences. He is the Fellow of IEEE and the Distinguished Member of CCF.

Wei Zhao (Fellow, IEEE) completed his undergraduate studies in physics at Shaanxi Normal University, China, in 1977, and received his MSc and PhD degrees in Computer and Information Sciences at the University of Massachusetts at Amherst in 1983 and 1986, respectively. Professor Zhao has served important leadership roles in academic including the Chief Research Officer at the American University of Sharjah, the Chair of Academic Council at CAS Shenzhen Institute of Advanced Technology, the eighth Rector of the University of Macau, the Dean of Science at Rensselaer Polytechnic Institute, the Director for the Division of Computer and Network Systems in the U.S. National Science Foundation, and the Senior Associate Vice President for Research at Texas A&M University. Professor Zhao has made significant contributions to cyber-physical systems, distributed computing, real-time systems, and computer networks. He led the effort to define the research agenda of and to create the very first funding program for cyber-physical systems in 2006. His research results have been adopted in the standard of Survivable Adaptable Fiber Optic Embedded Network. Professor Zhao was awarded the Lifelong Achievement Award by the Chinese Association of Science and Technology in 2005.

Yuan Wu (S'08-M'10-SM'16) received the PhD degree in Electronic and Computer Engineering from the Hong Kong University of Science and Technology in 2010. He is currently an Associate Professor with the State Key Laboratory of Internet of Things for Smart City, University of Macau and also with the Department of Computer and Information Science, University of Macau. During 2016-2017, he was a visiting scholar with Department of Electrical and Computer Engineering, University of Waterloo. His research interests include resource management for wireless networks, green communications and computing, mobile edge computing and edge intelligence. He was a recipient of the Best Paper Award from the IEEE International Conference on Communications in 2016, and the Best Paper Award from IEEE Technical Committee on Green Communications and Computing in 2017. Dr. Wu is currently on the Editorial Boards of IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, and IEEE INTERNET OF THINGS JOURNAL.

Authorized licensed use limited to: Queen Mary University of London. Downloaded on July 11,2024 at 14:10:40 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.See https://www.ieee.org/publications/rights/index.html for more information.