Online Layer-Aware Joint Request Scheduling, Container Placement, and Resource Provision in Edge Computing

Zhenzheng Li, Jiong Lou, *Member, IEEE*, Zhiqing Tang, *Member, IEEE*, Jianxiong Guo, *Member, IEEE*, Tian Wang, *Senior Member, IEEE*, Weijia Jia, *Fellow, IEEE*, and Wei Zhao, *Fellow, IEEE*

***Abstract*—Containers have emerged as a pivotal tool for service deployment in edge computing. Before running the container, an image composed of several layers must exist locally. Recent strategies have utilized layer-sharing in images to reduce deployment delays. However, existing research only focuses on a single aspect of container orchestration, like container placement, neglecting the joint optimization of the entire orchestration process. To fill in such gaps, this article introduces an online strategy that considers layer-aware container orchestration, encompassing request scheduling, container placement, and resource provision. The goal is to reduce costs, adapt to evolving user demands, and adhere to system constraints. We present an online optimization problem that accounts for various real-world factors in orchestration, including container and server expenses. An online algorithm is proposed, integrating a regularization-based approach and stepwise rounding to address this optimization problem efficiently. The regularization approach separates time-dependent container placement and server wake-up costs, requiring only current information and past decisions. The stepwise rounding process generates feasible solutions that meet system constraints, reducing computational costs. Additionally, a competitive ratio proof is provided for the proposed algorithm. Extensive evaluations demonstrate that our approach achieves about 20% performance enhancement compared to baseline algorithms.**

***Index Terms*—Container placement, request scheduling, resource provision, edge computing.**

Received 7 May 2024; revised 8 November 2024; accepted 15 November 2024. Date of publication 21 November 2024; date of current version 6 February 2025. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62272050 and Grant 62302048, in part by the Guangdong Key Lab of AI and Multi-modal Data Processing, United International College (UIC), Zhuhai under 2023-2024 Grants funded by the Guangdong Provincial Department of Education, in part by the Institute of Artificial Intelligence and Future Networks (BNU-Zhuhai) and Engineering Center of AI and Future Education, Guangdong Provincial Department of Science and Technology, China, in part by Zhuhai Science-Tech Innovation Bureau under Grant 2320004002772, and in part by the Interdisciplinary Intelligence Super Computer Center of Beijing Normal University at Zhuhai. *(Corresponding authors: Zhiqing Tang and Weijia Jia.)*

Zhenzheng Li is with the School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China, and also with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China (e-mail: zhenzhengli@mail.bnu.edu.cn).

Jiong Lou is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: lj1994@sjtu.edu.cn).

Zhiqing Tang and Tian Wang are with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China (e-mail: zhiqingtang@bnu.edu.cn; tianwang@bnu.edu.cn).

Jianxiong Guo and Weijia Jia are with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai 519087, China (e-mail: jianxiongguo@bnu.edu.cn; jiawj@bnu.edu.cn).

Wei Zhao is with the Shenzhen University of Advanced Technology, Shenzhen 518055, China (e-mail: zhao.wei@siat.ac.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TSC.2024.3504237, provided by the authors.

Digital Object Identifier 10.1109/TSC.2024.3504237

I. Introduction

EDGE computing leverages various clusters deployed at the edge of the network [1], thereby significantly enhancing the capabilities of the core network to support data-intensive and delay-sensitive applications [2], [3]. To enable efficient service deployment at the edge, container technology has emerged as a solution for hosting services [4], [5], [6], [7]. A container-based service bundles all essential components into its container image, which comprises layers representing changes to the file system, including additions, deletions, and modifications [8]. Upon receiving a user request, it is scheduled to the edge cluster, where the container is placed, and the necessary resources are provisioned to run the container. If the edge cluster lacks the locally stored layers of the container image, they must be downloaded from a remote cloud-based registry [9], [10]. *Hence, to deliver services to users through containers, container orchestration is essential, involving the following three main steps: 1) Scheduling user requests, 2) Placing containers in edge clusters, and 3) Provisioning resources in edge clusters.*

Most existing research on container orchestration primarily focuses on container placement. By taking into account the presence of images during container placement, various objectives can be accomplished, such as enhancing resource utilization [11], [12], [13], reducing service expenses [14], [15], [16], and speeding up startup [17]. Furthermore, exploring the layer-sharing nature of containers can further reduce costs [18]. Since multiple container images can share common base layers, it is possible to place several containers on the same edge cluster, allowing them to share common base layers and thereby accelerate deployment. Numerous existing studies address layer sharing aspects when downloading or placing container images [10], [19], [20]. Additionally, specific algorithms have been introduced to curtail system service deployment delay, storage usage, and request distribution expenses [21], [22], [23]. However, none of the existing research comprehensively encompasses the entire container orchestration process. In fact,

the scheduling of user requests impacts container placement, which in turn influences resource provisions. These three steps are intricately interconnected. *Therefore, optimizing the entire container orchestration process while considering layer sharing remains a critical issue that requires a solution.*

To optimize the costs of orchestration, we consider a layer-aware joint request scheduling, container placement, and resource provision problem. However, several challenges must be addressed. *The first challenge is how to consider and integrate different costs in container orchestration.* Scheduling at the layer granularity involves complex considerations due to the diversity of edge clusters, such as varying energy efficiency ratios, communication delays, and cluster states. Existing studies [10], [22], [23] focus on layer-based container placement but overlook real-world cost factors, potentially increasing system costs. Particularly, with layer sharing, containers sharing common layers are placed on a specific edge cluster, leading to a surge in requests to that cluster. This surge forces the cluster to activate servers for additional resources, causing significant wake-up delays [24]. Moreover, it may also result in scheduling requests to remote or low-efficiency edge clusters, leading to increased communication latency or higher energy consumption. Therefore, the system must trade off multiple costs, including resource provision and request scheduling, to jointly minimize overall costs when making layer-aware container placement decisions.

The second challenge is how to make online joint decisions while considering layer sharing and synthesis costs. In contrast to offline scenarios, where complete information such as future user requests is available, online scenarios necessitate decision-making based on dynamic and incomplete information [25]. This implies that online algorithms cannot wait for all the information to be known before making the decisions. Instead, decisions must be made incrementally as the problem unfolds, often leading to increased complexity of the problem. While some previous research relies on offline strategies for container placement based on prior knowledge of future user request patterns [21], [26], these methods do not account for the dynamic characteristics of online edge environments, such as user mobility and changing requirements [27]. Therefore, there is a critical necessity to develop online scheduling mechanisms that rely solely on current information.

In this paper, we explore the layer-aware joint decision-making problem focusing on overall costs for container orchestration in edge computing. Specifically, we formulate a Layer-aware Joint Request scheduling, Container placement, and Resource provision (LJRCR) problem, which aims to minimize the overall cost associated with request scheduling, container placement, storage, server energy, and wake-up. In LJRCR, container placement and resource provision are dynamic processes that require continuous adjustment, leading to time-dependent costs for both container placement and server wake-up. These time-dependent costs, where past decisions affect current choices and subsequently future outcomes, make solving LJRCR highly complex. The joint nature of LJRCR necessitates consideration of multiple interdependent objectives and precedence constraints. This interdependency complicates the process of finding an effective solution. Moreover, LJRCR is NP-Hard, which makes it challenging to solve even in offline scenarios.

To solve the LJRCR, we propose an Online Regularization and Rounding (ORR) algorithm. We utilize a regularization-based method to break down the time-dependent container placement and server wake-up costs, creating a series of time-independent convex subproblems. This decomposition allows us to calculate fractional solutions for jointly optimizing multiple interdependent decisions, utilizing only current information and past decisions. Subsequently, we develop a stepwise rounding process that, in conjunction with the regularization-based approach, refines the fractional solutions for each decision type in accordance with interdependent precedence constraints. For every decision type, ORR incrementally adjusts a pair of fractional solutions up and down in a compensatory manner. Our method involves a single rounding operation to meet all constraints, thus eliminating the need for repeated rounding attempts and reducing computational burdens. Additionally, we rigorously prove the competitive ratio of the ORR through theoretical analysis. Finally, we conduct a comprehensive evaluation of the ORR, considering various experimental setups and authentic data. The ORR demonstrates an approximate performance enhancement of 20% compared to baseline algorithms. The main contributions of this paper can be summarized as follows:

1) We formulate the LJRCR problem, which utilizes layer-sharing and addresses costs associated with request scheduling, container placement, and resource provision.
2) We present the ORR algorithm for the LJRCR, which decomposes the time-dependent costs into time-independent convex subproblems and employs a stepwise rounding process to derive feasible solutions.
3) We rigorously prove the competitive ratio of the ORR and assess it through extensive experiments, demonstrating a performance improvement of approximately 20% over baseline algorithms.

The remainder of this paper is organized as follows. Section II presents the related work. Section III introduces the system model and formulation of the LJRCR. Section IV presents the ORR. Section V analyzes and proves the competitive ratio of the ORR. Section VI provides the performance evaluation, and we conclude in Section VII.

II. Related Work

A. Container Placement

Container placement in edge computing poses a significant challenge due to the geographical dispersion and heterogeneity of edge clusters [28]. Sami et al. [29] propose an intelligent fog and service placement solution utilizing Deep Reinforcement Learning (DRL) to make proactive container placement decisions in advance of user requests, recognizing the necessity for swift and proactive service updates. Wang et al. [30] combine task scheduling with automatic scaling, presenting a delay-aware algorithm for task scheduling, container placement, and resource scaling. Zhang et al. [31] devise a joint task scheduling and containerization scheme that accounts for applications with interrelated tasks. Menouer et al. [32] introduce a multi-criteria

edge cluster selection for container placement, choosing the best cluster for each new container submission through a policy that balances multiple criteria related to user needs. However, these methods treat container images as a whole, overlooking the shared layers within them.

B. Layer-Aware Container Scheduling

Several studies show that leveraging the layer-sharing nature of container images can improve service deployment performance [33]. For example, Tang et al. [10] propose a layer-aware container scheduling algorithm that considers complex dependencies among layers and images, reducing overall task completion time. Lou et al. [20] introduce a layer-aware scheduling algorithm that integrates layer-sharing while jointly optimizing container allocation and layer download sequence to minimize total startup delay. Liu et al. [34] explore an optimal microservice placement strategy that balances layer and chain sharing to reduce microservice image pulling delays and communication costs. Given the complex inter-dependencies among multiple users, Tang et al. [19] propose a container migration algorithm to reduce overall migration cost. Gu et al. [26] explore the collaborative deployment of microservices by merging intra-server and inter-server layer-sharing to maximize edge throughput. However, most of these approaches either assume prior knowledge of future user request patterns or neglect the overall costs involved in layer-aware container orchestration decisions.

C. Joint Decision-Making

The joint optimization problem seeks to determine decision variables that simultaneously optimize multiple objectives or system components. Joint decision-making methods have demonstrated improved overall system performance in edge computing [35], [36]. Zhang et al. [37] approach the task offloading, content caching, and resource allocation as a Mixed Integer Non-Linear Programming (MINLP) problem to derive an optimal set of policies. Tran et al. [38] investigate the joint task offloading and resource allocation problem to maximize user gains. Yu et al. [39] address an optimization problem encompassing Unmanned Aerial Vehicle (UAV) location, communication, computational resource allocation, and task partitioning decisions. Furthermore, Khoramnejad et al. [40] focus on maximizing computed bits while minimizing energy consumption in joint resource allocation and task offloading decisions. Nan et al. [41] introduce optimization methods for task offloading decisions of vehicles and the allocation of uplink bandwidth and computing resources for a roadside device within its coverage area.

However, these studies do not address the layer-aware online joint decision-making problem, which considers the impact of various interrelated factors. To our knowledge, our work is the first to utilize layer-sharing to enhance container orchestration performance while fully accounting for system costs related to request scheduling, container placement, and resource provision.

TABLE I
MAIN NOTATIONS

Notations	Description
$\mathcal{N}$	Set of geographically dispersed edge clusters
$\mathcal{T}$	Set of time slots
$\mathcal{I}$	Set of container images
$\mathcal{L}$	Set of layers
h_i^l	Whether container image i contains layer l (i.e., $h_i^l = 1$) or not (i.e., $h_i^l = 0$)
D_n	Maximum number of servers on the edge cluster n
Q_n	Request processing capability for each server in the edge cluster n
c_n^l	Weighted storage cost factor
$c_{nn'}^i$	Weighted request scheduling cost factor
c_n	Weighted server energy cost factor
d_n^l	Weighted container placement cost factor
d_n	Weighted server wake-up cost factor
$r_n^i(t)$	Number of user requests i arriving at edge cluster n
$x_n^l(t)$	Whether the edge cluster n stores layer l (i.e., $x_n^l(t) = 1$) or not (i.e., $x_n^l(t) = 0$)
$y_{nn'}^i(t)$	Number of user requests i arriving at edge cluster n to be scheduled to edge cluster n'
$z_n(t)$	Number of servers activated on the edge cluster n

III. System Model and Problem Formulation

We consider a generalized container orchestration scenario in edge computing, as illustrated in Fig. 1. When a user sends a request, the edge cluster can schedule the request to any cluster (*Request scheduling*). Then, the cluster should download the required layers from the registry on the remote cloud to construct a complete image and start up the container that provides the service [22] (*Container placement*). When the currently activated servers cannot handle incoming requests, the sleeping servers are woken up to provide the necessary computing resources [24] (*Resource provision*). In this section, we formulate the layer-aware joint optimization problem, which aims to find a set of decision variables that optimize multiple objectives or system components concurrently. Specifically, this paper leverages layer-sharing of multiple container images to jointly minimize the system costs associated with request scheduling, container placement, storage, server energy, and wake-up. The main notations are summarized in Table I.

A. System Model

Edge clusters: Edge computing networks consist of a set of geographically dispersed edge clusters, denoted as $\mathcal{N} = \{1, 2, \ldots, |\mathcal{N}|\}$. Each edge cluster comprises two main components: the storage, responsible for storing data, and the servers, responsible for processing user requests [42]. Let $\mathcal{T} = \{1, 2, \ldots, |\mathcal{T}|\}$ denote the set of time slots, and the time slot is assumed as a fixed interval of time. A container can only service one request in a time slot. These edge clusters can communicate with each other and work collaboratively. The central controller makes control decisions during each time slot. The edge clusters receive these control decisions, execute the corresponding actions, and transmit the cluster's state back to the central controller for decision-making in the next time slot.

Containers and images: A container image registry on the remote cloud stores a set of container images $\mathcal{I} = \{1, 2, \ldots, |\mathcal{I}|\}$.

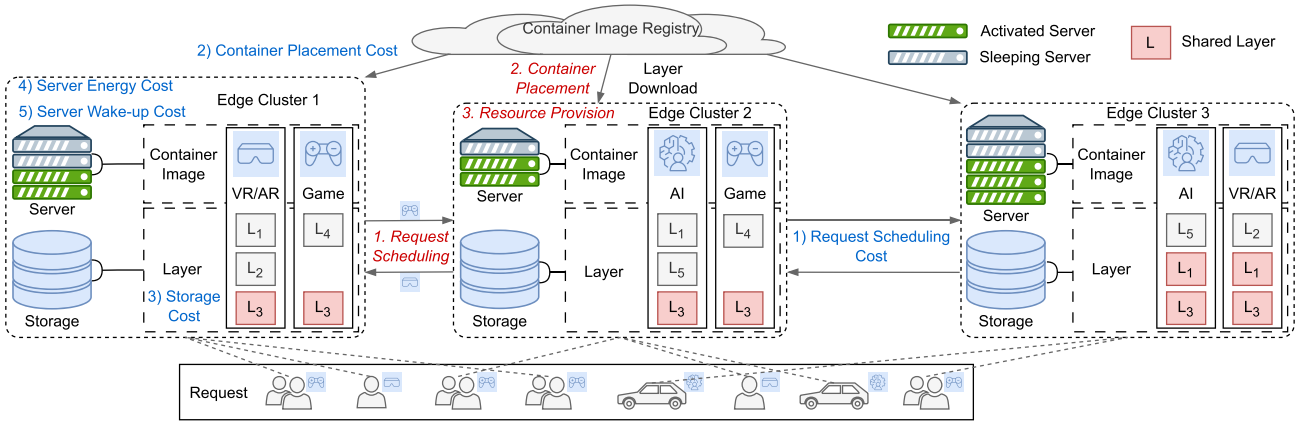


Fig. 1. Overview of container orchestration in edge computing, including request scheduling, container placement, and resource provision. Container images can share multiple base layers, such as L_1 and L_3. Service deployment can be significantly accelerated by placing containers with shared layers within the same cluster. This paper leverages layer-sharing of multiple container images to jointly minimize request scheduling, container placement, storage, server energy, and wake-up costs.

Each container image $i \in \mathcal{I}$ consists of multiple layers. The notation $\mathcal{L} = \{1, 2, \ldots, |\mathcal{L}|\}$ is used to denote the set of layers. We use $h_i^l \in \{0, 1\}$ to indicates whether the container image i contains layer $l \in \mathcal{L}$ (i.e., $h_i^l = 1$) or not (i.e., $h_i^l = 0$). To deploy a container i on a cluster $n \in \mathcal{N}$, the cluster n must store all the layers required by the container image i. Otherwise, it must download the absent layers from the container image registry. Consequently, if two container images share a common layer l, colocating both images within the same cluster enables layer l to be downloaded only once, thereby reducing redundant downloads and accelerating service deployment.

User requests: Let $r_n^i(t) \in \mathbb{N}$ represent the number of user requests i arriving at the cluster n during time slot t. Since the user request i can be served by the corresponding container i in the edge cluster, we use the notation i in a slightly abusive way. The requests can be scheduled to any cluster n'. Let $y_{nn'}^i(t) \in \mathbb{N}$ represent the number of user requests i arriving at the cluster n to be scheduled to the cluster n'. We use $x_n^l(t) \in \{0, 1\}$ to indicate whether the cluster n stores layer l (i.e., $x_n^l(t) = 1$) or not (i.e., $x_n^l(t) = 0$). Thus, the relationship between $x_n^l(t)$ and $y_{nn'}^i(t)$ can be expressed as $y_{nn'}^i(t) h_i^l \leq x_{n'}^l(t) r_n^i(t)$. This inequality indicates that the cluster n' has all the layers needed to deploy the container image i and that the number of requests scheduled to cluster n' cannot exceed the number of requests received by the source cluster n. Scheduling each user request to an edge cluster is essential for stable service delivery. Therefore, it should hold that $r_n^i(t) \leq \sum_{n' \in \mathcal{N}} y_{nn'}^i(t)$.

The storage in the edge cluster serves to store all layers, which the server within the cluster can access through the internal high-speed local area network. Let $z_n(t) \in \mathbb{N}^+$ denote the number of servers activated on the cluster n. Note that $z_n(t)$ cannot exceed the maximum number of servers D_n on the cluster n. Furthermore, let Q_n be the request processing capability (i.e., the number of containers that can be run) related to the computing capability for each server in the cluster n. Hence, the product of D_n and Q_n represent the overall computing resources at the cluster n, while the product of $z_n(t)$ and Q_n indicates the currently provisioned computing resources at the cluster n. The cluster must ensure the provision of sufficient computing resources to process incoming scheduled requests. Therefore, the relationship between $y_{nn'}^i(t)$ and $z_n(t)$ can be expressed as $\sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} y_{nn'}^i(t) \leq Q_{n'} z_{n'}(t)$.

B. System Cost

The total system cost involved in container orchestration consists of the following components. The cost of request scheduling during *request scheduling*; the cost of container placement and storage during *container placement*; and the cost of server energy and wake-up during *resource provision*.

Request scheduling cost: It is proportional to the service delay of the request scheduling. Let $H_{nn'}$ represent the communication delay [43] between clusters n and n', H_{nr} represent the communication delay between cluster n and the container image registry, and b_i indicate the data size of user request i, and B_n be the bandwidth of the cluster n. Then, the total request scheduling cost is defined as $\sum_{t \in \mathcal{T}} \sum_{n' \in \mathcal{N}} \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} c_{nn'}^i y_{nn'}^i(t)$. Here, $c_{nn'}^i$ represents the weighted request scheduling cost factor, which quantifies the communication cost associated with scheduling request i from cluster n to cluster n'. Specifically, $c_{nn'}^i = W_2(H_{nn'} + \frac{b_i}{B_n})$, where W_2 is the weight.

Container placement cost: It is proportional to the deployment delay for placing containers. An edge cluster must download the required layers from the image registry to create a complete container image, incurring additional initialization delay. The delay for the cluster n to download layer l from the image registry is $H_{nr} + \frac{p_l}{B_n}$. The total container placement cost is then obtained by $\sum_{t \in \mathcal{T}} \sum_{n \in \mathcal{N}} \sum_{l \in \mathcal{L}} d_n^l [x_n^l(t) - x_n^l(t-1)]^+$, where $[x]^+ = \max\{x, 0\}$. d_n^l represents the weighted container placement cost factor, which quantifies the delay cost associated with downloading the missing layer l to the cluster n. Specifically, $d_n^l = W_3(H_{nr} + \frac{p_l}{B_n})$, where W_2 is the weight. Our container placement cost model differs from prior research on layer-aware container placement [21], [22] by explicitly considering the dynamics of online scenarios. Since the decision at the current time t is influenced by the prior decision $x_n^l(t-1)$, and the

decisions made at t will impact future decisions, the container placement cost exhibits time-dependent characteristics.

Storage cost: It is the storage consumption of the stored layers during container placement. Let p_l represent the storage space occupied by layer l. Then, the total storage cost is obtained by $\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} c_n^l x_n^l(t)$. Here, c_n^l indicates the weighted storage cost factor, calculated as $c_n^l = W_1 p_l$, where W_1 is the weight.

Server energy cost: Sustaining active status for servers within edge clusters results in energy consumption. Let c_n represent the weighted server energy cost factor for the cluster n. Then, the total server energy cost is given by $\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} c_n z_n(t)$.

Server wake-up cost: It is proportional to the additional delay incurred when waking up a sleeping server. The sleeping servers will be woken up when the currently provisioned computing resources are inadequate. Let d_n represent the weighted server wake-up cost factor, which qualifies the wake-up delay. The total server wake-up cost is calculated as $\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} d_n[z_n(t) - z_n(t-1)]^+$. Similar to the container placement cost, the server wake-up cost also demonstrates time-dependent characteristics.

Three of these costs (request scheduling, storage, and server energy costs) are linear and associated with each independent time slot. The remaining two costs are switching costs, which include container placement and server wake-up costs for each pair of consecutive time slots.

C. Problem Formulation

We formulate a layer-aware joint request scheduling, container placement, and resource provision problem $\mathbf{P}$, aiming to minimize the system costs of five components over time while satisfying the time-varying user requests and respecting various system constraints. The problem $\mathbf{P}$ is as follows:

$$\begin{aligned}
\mathbf{P}: \quad & \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} c_n^l x_n^l(t) + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} c_n z_n(t) \\
& + \sum_{t\in\mathcal{T}}\sum_{n'\in\mathcal{N}}\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} c_{nn'}^i y_{nn'}^i(t) \\
& + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} d_n^l \left[x_n^l(t) - x_n^l(t-1)\right]^+ \\
& + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} d_n \left[z_n(t) - z_n(t-1)\right]^+ \qquad (1)
\end{aligned}$$

$$\text{s.t.} \quad y_{nn'}^i(t) h_i^l \le x_{n'}^l(t) r_n^i(t), \qquad \forall l, \forall i, \forall n, \forall n', \forall t \quad (1a)$$

$$r_n^i(t) \le \sum_{n'\in\mathcal{N}} y_{nn'}^i(t), \qquad \forall i, \forall n, \forall t \quad (1b)$$

$$\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} y_{nn'}^i(t) \le Q_{n'} z_{n'}(t), \quad \forall n', \forall t \quad (1c)$$

$$x_n^l(t) \in \{0,1\}, \qquad \forall l, \forall i, \forall n, \forall t \quad (1d)$$

$$y_{nn'}^i(t) \in \{0,1,\ldots\}, \qquad \forall i, \forall n, \forall n', \forall t \quad (1e)$$

$$z_n(t) \in \{1,2,\ldots,D_n\}, \qquad \forall n, \forall t \quad (1f)$$

where $\forall l, \forall i, \forall n, \forall n', \forall t$ represents $\forall l \in \mathcal{L}, \forall i \in \mathcal{I}, \forall n \in \mathcal{N}, \forall n' \in \mathcal{N}, \forall t \in \mathcal{T}$, respectively. Constraint (1a) ensures that the cluster n' has all the layers required for deploying container image i and the number of requests scheduled to the cluster n' cannot exceed the number of requests received by the source cluster n. Constraint (1b) guarantees that each user request is scheduled to an edge cluster. Constraint (1c) ensures that the cluster provisions enough computing resources to handle the scheduled requests.

IV. Online Algorithm Design

In this section, we derive the decision by solving the problem $\mathbf{P}$. However, optimizing $\mathbf{P}$ is non-trivial due to the following primary challenges. First, even in the offline case where $r_n^i(t), \forall t$ is known at the time of decision-making, if we simplify the problem $\mathbf{P}$ by overlooking the switching costs, our problem is an advanced covering problem [44], a class recognized for its NP-Hardness and can only be solved heuristically. In the online scenario, decision-making at time t relies solely on the current information $r_n^i(t)$ and previous decisions $x_n^l(t-1)$ and $z_n(t-1)$, which further increases the complexity of the problem. Second, the algorithm design is further complicated by the interdependent precedence constraints and time-dependence arising from the container placement and server wake-up costs. These challenges pose difficulties in leveraging the existing information for long-term cost optimization and constraints satisfy.

Conventional randomized rounding approaches hinge on the relaxation of programming problems and leverage convex optimization techniques to attain fractional solutions [45]. Typically, these approaches are limited in their applicability to individual time slots, rendering them incapable of simultaneously addressing the time-dependent switching costs for long-term cost optimization in this context. Moreover, the conventional randomized rounding approach demands repeated rounding tries to guarantee the satisfaction of all interdependent precedence constraints, a process characterized by considerable time consumption. To address this challenge, we aim to develop an efficient online algorithm capable of achieving both provable theoretical performance and polynomial time complexity. We summarize the proposed ORR in Algorithm 1 and provide a roadmap for the ORR in Fig. 2. As shown in the blue part of Fig. 2, we relax, regularize, and decompose the problem $\mathbf{P}$ to obtain the convex subproblem $\mathbf{P}_2^t$. Thus, $\mathbf{P}_2^t$ are time-independent, and we can solve $\mathbf{P}_2^t$ by invoking Online Regularization-based Approach (ORA) based on current information and previous decisions. In the orange part of Fig. 2, a stepwise rounding process is designed to obtain the feasible integral solutions. The stepwise rounding process entails a single rounding process to satisfy all constraints. We prove that the ORR satisfies all constraints, thereby resolving the issue of necessitating repeated rounding tries. Next, we will detail the algorithm design.

A. Online Regularization-Based Approach

Before introducing the specific algorithm, it is instructive to examine the structure of the problem $\mathbf{P}$. The objective function (1) comprises three linear terms and two switching terms, exhibiting evident non-convex characteristics. Additionally, the

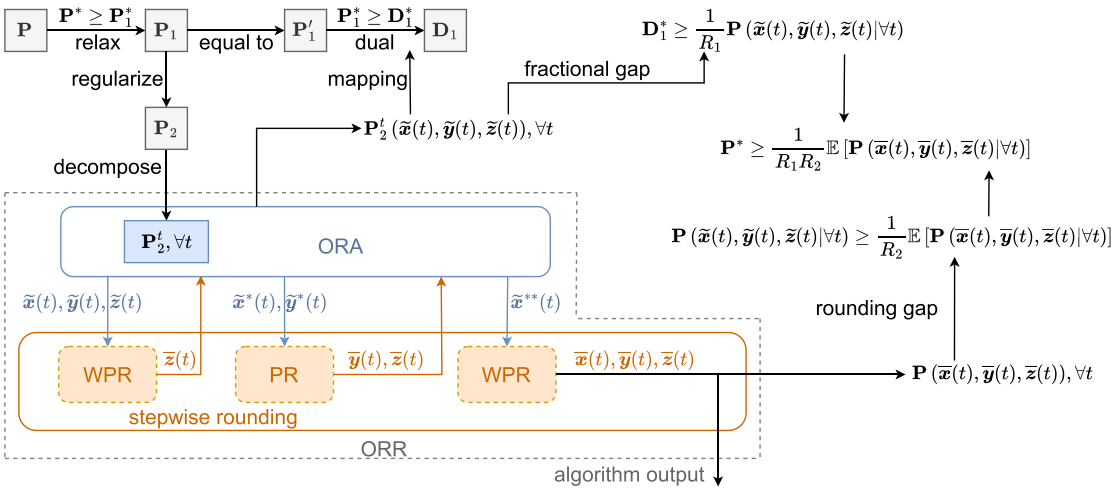


Fig. 2. A roadmap of the ORR (blue and orange) and performance analysis (black).

constraints encompass three linear precedence constraints (1a)–(1c), along with integral constraints (1d)–(1f). We should transform the switching term into a convex function and relax the integral variable, and then we can formulate a convex optimization problem to obtain fractional solutions.

According to this idea, we first obtain the relaxed problem $\mathbf{P_1}$ by relaxing the integral variables $\boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{z}(t), \forall t$ of the problem $\mathbf{P}$. The remaining challenge in solving the relaxed problem $\mathbf{P_1}$ is converting the switching costs, i.e., container placement and server wake-up costs. To this end, we employ the relative entropy function [46] to replace the container placement and server wake-up costs in our relaxed problem $\mathbf{P_1}$, obtaining a regularized problem $\mathbf{P_2}$. The relative entropy function, a proven convex function, has been widely used to approximate the L1-distance term. We express regularized problem as the sum of subproblems, $\mathbf{P_2} = \sum_{t\in\mathcal{T}} \mathbf{P}_2^t$, illustrated as follows:

$$\begin{aligned}\mathbf{P}_2^t = & \sum_{n'\in\mathcal{N}}\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} c_{nn'}^i y_{nn'}^i(t) \\ & + \sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} c_n^l x_n^l(t) + \sum_{n\in\mathcal{N}} c_n z_n(t) \\ & + \sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} \frac{d_n^l}{\eta_x}\left(\left(x_n^l(t)+\varepsilon\right)\ln\frac{x_n^l(t)+\varepsilon}{x_n^l(t-1)+\varepsilon} - x_n^l(t)\right) \\ & + \sum_{n\in\mathcal{N}} \frac{d_n}{\eta_z}\left(\left(z_n(t)+\varepsilon'\right)\ln\frac{z_n(t)+\varepsilon'}{z_n(t-1)+\varepsilon'} - z_n(t)\right) \quad (2)\end{aligned}$$

s.t. (1a) - (1c)

$$x_n^l(t) \in [0,1], \qquad \forall l, \forall n \qquad (2a)$$

$$y_{nn'}^i(t) \geq 0, \qquad \forall i, \forall n, \forall n' \qquad (2b)$$

$$z_n(t) \in [1, D_n], \qquad \forall n \qquad (2c)$$

where $\eta_x = \ln(1+\frac{1}{\varepsilon})$ and $\eta_z = \ln(1+\frac{1}{\varepsilon'})$. To prevent the occurrence of exception values, we add small positive constants ε and ε' to both the denominator and numerator of the fraction in the relative entropy function. Moreover, we multiply the relative entropy function by η_x and η_z to normalize the regularized container placement and server wake-up costs, respectively.

Since the objective function (2) is convex and all constraints are linear, the subproblem $\mathbf{P}_2^t$ is a convex problem for each time slot t. Moreover, the subproblem $\mathbf{P}_2^t$ requires only the current information $r_n^i(t)$ and previous decisions $x_n^l(t-1)$ and $z_n(t-1)$ as input parameters in time slot t. Consequently, the subproblem $\mathbf{P}_2^t$ can be efficiently solved by the standard convex optimization algorithms (e.g., interior point method [47]) within polynomial time complexity. This implies that fractional solution $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$ can be derived by addressing problem $\mathbf{P}_2^t$ individually for each time slot, so the regularized problem $\mathbf{P_2}$ can be solved as a series of time-independent convex subproblems $\mathbf{P}_2^t$ to obtain the optimal solution. Formally, as in line 4 of the ORR, we invoke the subroutine ORA to obtain the fractional solutions $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$. ORA generates the fractional solutions by the interior point method within each time slot.

B. Rounding Process

Given the integral constraints (1d)–(1f) inherent in the problem $\mathbf{P}$, it is essential to note that the fractional solutions $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$ does not satisfy to these constraints. Hence, it is necessary to round the fractional solutions to the feasible integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$. Conventional randomized rounding approaches [45], commonly employed for converting fractional solutions into integral counterparts, encounter particular challenges when applied to our specific problem. It is not assured that these approaches can reliably meet the conditions specified by constraints (1b) and (1c). Therefore, repeated rounding tries are mandated for the traditional randomized rounding algorithm to satisfy both constraints. This iterative process introduces a considerable computational overhead.

A more formidable challenge arises in the precedence constraint (1a). When utilizing the conventional randomized rounding approaches, there is a possibility that particular layers associated with a specific container image may be rounded up while others are simultaneously rounded down. This situation could

Algorithm 1: ORR.

Input: $\mathcal{N}, \mathcal{I}, \mathcal{L}, \boldsymbol{h}, \boldsymbol{D}, \boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{d}, \varepsilon, \varepsilon'$
Output: Integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t), \forall t$
1: **Initialize** $\boldsymbol{x} = 0, \boldsymbol{y} = 0, \boldsymbol{z} = 0$
2: **for** $t \in \mathcal{T}$ **do**
3: Get user requests $\boldsymbol{r}(t)$
4: Invoke ORA to obtain the fractional solutions $\widetilde{\boldsymbol{x}}(t)$, $\widetilde{\boldsymbol{y}}(t)$ and $\widetilde{\boldsymbol{z}}(t)$
5: Integral solutions $\overline{\boldsymbol{z}}(t) = \text{WPR}(\widetilde{\boldsymbol{z}}(t), \boldsymbol{Q})$
6: Substitute coefficients c_n^l and d_n^l of objective function (2) with $c_n^l Q_n$ and $d_n^l Q_n$, respectively, fix $\overline{\boldsymbol{z}}(t)$ and invoke ORA to obtain the fractional solutions $\widetilde{\boldsymbol{x}}^*(t), \widetilde{\boldsymbol{y}}^*(t)$
7: Integral solutions $\overline{\boldsymbol{y}}(t) = \text{PR}(\widetilde{\boldsymbol{y}}^*(t))$
8: Fix $\overline{\boldsymbol{y}}(t)$ and $\overline{\boldsymbol{z}}(t)$, replace constraint (1a) with $u(0 < \overline{y}_{nn'}^i(t) h_i^l) \le x_{n'}^l(t)$, and invoke ORA to obtain the fractional solutions $\widetilde{\boldsymbol{x}}^{**}(t)$
9: Integral solutions $\overline{\boldsymbol{x}}(t) = \text{WPR}(\widetilde{\boldsymbol{x}}^{**}(t), \boldsymbol{c})$
10: **return** $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$

Subroutine ORA: Online Regularization-Based Approach.

Input:
$\mathcal{N}, \mathcal{I}, \mathcal{L}, \boldsymbol{h}, \boldsymbol{D}, \boldsymbol{Q}, \boldsymbol{c}, \boldsymbol{d}, \varepsilon, \varepsilon', t, \boldsymbol{x}(t-1), \boldsymbol{z}(t-1), \boldsymbol{r}(t)$
Output: $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$
1: Invoke the interior point method to solve $\mathbf{P}_2^t$:
min: $\sum_{n' \in \mathcal{N}} \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} c_{nn'}^i y_{nn'}^i(t)$
$+ \sum_{n \in \mathcal{N}} \sum_{l \in \mathcal{L}} c_n^l x_n^l(t) + \sum_{n \in \mathcal{N}} c_n z_n(t)$
$+ \sum_{n \in \mathcal{N}} \sum_{l \in \mathcal{L}} \frac{d_n^l}{\eta_x} ((x_n^l(t) + \varepsilon) \ln \frac{x_n^l(t) + \varepsilon}{x_n^l(t-1) + \varepsilon} - x_n^l(t))$
$+ \sum_{n \in \mathcal{N}} \frac{d_n}{\eta_z} ((z_n(t) + \varepsilon') \ln \frac{z_n(t) + \varepsilon'}{z_n(t-1) + \varepsilon'} - z_n(t))$
s.t. (1a) – (1a), (2a) – (2c)
2: **return** $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$

Subroutine WPR: Weighted Pairwise Rounding.

Input: $\widetilde{\boldsymbol{u}}(t), \boldsymbol{W}$
Output: $\overline{\boldsymbol{u}}(t)$
1: If input is $\widetilde{\boldsymbol{z}}(t)$, set $\mathcal{J} = \emptyset, \mathcal{K} = \mathcal{N}, \boldsymbol{W} = \boldsymbol{Q}$
2: If input is $\widetilde{\boldsymbol{x}}(t)$, set $\mathcal{J} = \mathcal{N}, \mathcal{K} = \mathcal{L}, \boldsymbol{W} = \boldsymbol{c}$
3: **for** $j \in \mathcal{J}$ **do**
4: **Initialize** $\widetilde{\mathcal{K}} = \mathcal{K}$
5: **for** $k \in \mathcal{K}$ **do**
6: $\mu_j^k = \widetilde{u}_j^k(t) - \lfloor \widetilde{u}_j^k(t) \rfloor$
7: $\widetilde{\mathcal{K}} = \widetilde{\mathcal{K}} - \{k | \mu_j^k \in \{0, 1\}\}$
8: **while** $|\widetilde{\mathcal{K}}| > 1$ **do**
9: Randomly select $k_1, k_2 \in \widetilde{\mathcal{K}}$, where$k_1 \neq k_2$
10: $\omega_1 = \min\{1 - \mu_j^{k_1}, \frac{W_j^{k_2}}{W_j^{k_1}} \mu_j^{k_2}\}$
11: $\omega_2 = \min\{\mu_j^{k_1}, \frac{W_j^{k_2}}{W_j^{k_1}} (1 - \mu_j^{k_2})\}$
12: With the probability $\frac{\omega_2}{\omega_1 + \omega_2}$ set $\mu_j^{k_1} = \mu_j^{k_1} + \omega_1, \mu_j^{k_2} = \mu_j^{k_2} - \frac{W_j^{k_1}}{W_j^{k_2}} \omega_1$
13: With the probability $\frac{\omega_1}{\omega_1 + \omega_2}$ set $\mu_j^{k_1} = \mu_j^{k_1} - \omega_2, \mu_j^{k_2} = \mu_j^{k_2} + \frac{W_j^{k_1}}{W_j^{k_2}} \omega_2$
14: **if** $\mu_j^{k_1} \in \{0, 1\}$ **then**
15: Set $\overline{u}_j^{k_1}(t) = \lfloor \widetilde{u}_j^{k_1}(t) \rfloor + \mu_j^{k_1}, \widetilde{\mathcal{K}} = \widetilde{\mathcal{K}} - \{k_1\}$
16: **if** $\mu_j^{k_2} \in \{0, 1\}$ **then**
17: Set $\overline{u}_j^{k_2}(t) = \lfloor \widetilde{u}_j^{k_2}(t) \rfloor + \mu_j^{k_2}, \widetilde{\mathcal{K}} = \widetilde{\mathcal{K}} - \{k_2\}$
18: **if** $|\widetilde{\mathcal{K}}| = 1$ **then**
19: For the only one element $k \in \widetilde{\mathcal{K}}$ set $\overline{u}_j^k(t) = \lceil \widetilde{u}_j^k(t) \rceil, \widetilde{\mathcal{K}} = \widetilde{\mathcal{K}} - \{k\}$
20: **return** $\overline{\boldsymbol{u}}(t)$

potentially lead to the catastrophic outcome of failing to acquire the essential layers required to construct a complete container image. Motivated by the abovementioned challenges, we design a stepwise rounding process, as illustrated in lines 5 to 9 of the ORR. Subsequently, we will progressively expound the details of the stepwise rounding process.

1) Rounding $\widetilde{\boldsymbol{z}}(t)$: If we choose to round the variables other than $\widetilde{\boldsymbol{z}}(t)$ first, there is a potential risk of not consistently achieving a feasible $\overline{\boldsymbol{z}}(t)$, primarily due to the constraints (1c) and (1f). Based on the above reason, we first round $\widetilde{\boldsymbol{z}}(t)$ to obtain the integral solutions $\overline{\boldsymbol{z}}(t)$ as depicted in line 5 of the ORR. Conservative rounding policies, which round up all decision variables, can lead to high costs. On the other hand, radical rounding strategies, which round down all decision variables, can result in the system being unable to ensure sufficient resources to process tasks. To address this issue, we develop the subroutine Weighted Pairwise Rounding (WPR), inspired by the dependent rounding technique [48], at every time slot to convert the fractional solutions $\widetilde{\boldsymbol{z}}(t)$ to the integral solutions $\overline{\boldsymbol{z}}(t)$.

The main idea of the WPR involves selecting two float solutions and conducting rounding in a complementary manner. In WPR, a weight is assigned to the rounding according to the request processing capacity Q_n of the cluster n, as shown in line 1. In lines 4 to 7, we find the set of float solutions in $\widetilde{\boldsymbol{z}}(t)$. As long as the number of elements in the set $\widetilde{\mathcal{K}}$ is more than one, WPR randomly selects two elements from the set $\widetilde{\mathcal{K}}$. In lines 10 to 11, it sets the rounding factor as the product of the probability and the rounding weight. The probability values of these two elements are subsequently updated randomly in lines 12 to 13. As shown in lines 14 to 17, if the probability becomes 0 (or 1), it will be rounded down (or rounded up). During the update process, one element undergoes an increase while the other decreases. The complementary strategy can minimize the gap from the original cost. It is important to note that in each round of iteration, the elements of the set $\widetilde{\mathcal{K}}$ are reduced by at least one. In lines 18 to 19, if $\widetilde{\mathcal{K}}$ has only one element, WPR directly rounds it up to ensure sufficient computing resources for processing incoming requests.

2) Rounding $\widetilde{\boldsymbol{y}}(t)$ and $\widetilde{\boldsymbol{x}}(t)$: Upon the rounding of $\widetilde{\boldsymbol{z}}(t)$, it is possible that some of the rounded values may undergo a round down, resulting in certain clusters being unable to accommodate the scheduled requests $\widetilde{\boldsymbol{y}}(t)$. This outcome could lead to the fractional solutions $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)$ becoming infeasible. To this end, we fix the integral solutions $\overline{\boldsymbol{z}}(t)$ and re-invoke the ORA to obtain

Subroutine PR: Pairwise Rounding.

Input: $\widetilde{\boldsymbol{y}}(t)$
Output: $\overline{\boldsymbol{y}}(t)$
1: **for** $n \in \mathcal{N}$ **do**
2: Construct a bipartite graph $G = (V_{n'}, V_i, E)$
3: For $(V_{n'}, V_i) \in E$, its value is $\widetilde{y}^i_{nn'}(t)$
4: **repeat**
5: Construct subgraph $(V_{n'}, V_i, F)$ by taking the float edges of $(V_{n'}, V_i, E)$
6: Search a simple cycle or maximal path in the subgraph $(V_{n'}, V_i, F)$ via Depth-First-Search
7: Divide cycle/path into two matching M_1, M_2 respectively
8: Set $\lambda^i_{nn'} = \widetilde{y}^i_{nn'}(t) - \lfloor \widetilde{y}^i_{nn'}(t) \rfloor$
9: Set $\omega_1 = \min\{\gamma : (\exists (n', i) \in M_1 : \lambda^i_{nn'} + \gamma = 1) \bigvee (\exists (n', i) \in M_2 : \lambda^i_{nn'} - \gamma = 0)\}$
10: Set $\omega_2 = \min\{\gamma : (\exists (n', i) \in M_1 : \lambda^i_{nn'} - \gamma = 0) \bigvee (\exists (n', i) \in M_2 : \lambda^i_{nn'} + \gamma = 1)\}$
11: With the probability $\frac{\omega_2}{\omega_1+\omega_2}$ set $\widetilde{y}^i_{nn'}(t) = \widetilde{y}^i_{nn'}(t) + \omega_1, \forall (n', i) \in M_1$ $\widetilde{y}^i_{nn'}(t) = \widetilde{y}^i_{nn'}(t) - \omega_1, \forall (n', i) \in M_2$
12: With the probability $\frac{\omega_1}{\omega_1+\omega_2}$ set $\widetilde{y}^i_{nn'}(t) = \widetilde{y}^i_{nn'}(t) - \omega_2, \forall (n', i) \in M_1$ $\widetilde{y}^i_{nn'}(t) = \widetilde{y}^i_{nn'}(t) + \omega_2, \forall (n', i) \in M_2$
13: **until** $F = \emptyset$
14: **return** $\overline{\boldsymbol{y}}(t)$

the feasible fractional solutions. Before proceeding further, it has come to our attention that the value of real $x^l_{n'}(t)$ cannot be tightly bounded by (1a) as the value of real $y^i_{nn'}(t)$ approaches 0. This scenario may increase the overall cost, as it underestimates the storage and container placement costs associated with $x^l_{n'}(t)$ on a specific cluster where sporadic requests are scheduled to that cluster. To alleviate this issue, within the objective function (2), we substitute coefficients c^l_n and d^l_n with $c^l_n Q_n$ and $d^l_n Q_n$, respectively, as demonstrated in the line 6 of the ORR. By employing this trick, we can alleviate the underestimation of storage and container placement costs compared to the request scheduling costs, thus mitigating this phenomenon. Following the re-invocation of ORA to obtain fractional $\widetilde{\boldsymbol{y}}^*(t)$, the integral solutions $\overline{\boldsymbol{y}}(t)$ are based on $\widetilde{\boldsymbol{y}}^*(t)$, as shown in the line 7 of the ORR.

Rounding $\widetilde{\boldsymbol{y}}^*(t)$ down directly may cause some requests to fail (i.e., violate constraint (1b)), while rounding $\widetilde{\boldsymbol{y}}^*(t)$ up directly may lead to insufficient computing resources on specific edge clusters (i.e., violate constraint (1c)). To this end, we develop the subroutine Pairwise Rounding (PR) subroutine, inspired by the dependent rounding technique [48], which involves selecting two float solutions and conducting rounding in a complementary manner. PR constructs the bipartite graph G and extracts a subgraph F with float values for the edges of G. For $\forall n$, the nodes of the bipartite graph G represent the user requests and the edge clusters, respectively. The edge between two nodes represents the assigned value of the request scheduling. In lines 6 to 7, the Depth-First-Search is used to find a simple cycle or maximal path in F. It then divides the cycle or path into two matchings, M_1 and M_2. In lines 9 to 10, we identify the smallest Euclidean distance between rounds up or down in the matchings. Moreover, the values of the float edges in the matchings are randomly updated in lines 11 to 12. Note that an increase in the float edge within the matching M_1 corresponds to an increase in the assigned value for request scheduling, accompanied by a corresponding reduction in the float edge within the matching M_2. This approach ensures that constraints are satisfied while preventing a significant cost increase. The inner loop is iterated until the subgraph F becomes empty.

Similar to obtaining $\overline{\boldsymbol{y}}(t)$, a three-step procedure is employed to obtain the integral solutions $\overline{\boldsymbol{x}}(t)$. First, after rounding $\widetilde{\boldsymbol{z}}(t)$ and $\widetilde{\boldsymbol{y}}^*(t)$, $\overline{\boldsymbol{y}}(t)$ and $\overline{\boldsymbol{z}}(t)$ are fixed in the line 8 of the ORR. To ensure the correct deployment of all layers necessary for constructing a complete container image, we substitute constraint (1a) with the following expression: $u(0 < \overline{y}^i_{nn'}(t) h^l_i) \leq x^l_{n'}(t)$, where the step function $u(x)$ is defined as 1 for $x > 0$ and 0 otherwise. This constraint guarantees properly downloading all necessary layers to the corresponding edge clusters. Since $\overline{\boldsymbol{y}}(t)$ is a fixed constant, the behavior of the replacing constraint does not affect the linear nature of the constraint, thus addresses the challenges posed by constraint (1a). Subsequently, we invoke ORA again to obtain the fractional solutions $\widetilde{\boldsymbol{x}}^{**}(t)$. Finally, WPR is invoked to round $\widetilde{\boldsymbol{x}}^{**}(t)$ to obtain the integral solutions $\overline{\boldsymbol{x}}(t)$, as shown in the line 9 of the ORR.

V. Theoretical Performance Analysis

In this section, we evaluate the feasibility of the final integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$ and the time complexity of the ORR algorithm. We also present a rigorous proof of the theoretical performance of the ORR regarding its competitive ratio. The competitive ratio R evaluates the worst-case performance of the ORR compared to the optimal offline solution, as expressed by $\mathbb{E}[\mathbf{P}(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t) | \forall t)] \leq R\mathbf{P}^*$, where $\mathbb{E}[\mathbf{P}(\cdot)]$ represents the expected objective value of problem $\mathbf{P}$ at the evaluation point, and $\mathbf{P}^*$ denotes the optimal objective value of the offline solution.

For ease of reading, we provide a roadmap for the performance analysis as the black part of Fig. 2. First, we introduce the concept of the equivalent relaxation auxiliary problem $\mathbf{P}'_1$ and its corresponding dual problem, serving as the critical bridge connecting the problem $\mathbf{P}$ with the regularized problem $\mathbf{P}_2$. We establish a mapping to generate a feasible solution for the dual problem. Following this, we employ the Karush-Kuhn-Tucker (KKT) conditions [49] as a pivotal analytical tool to scrutinize the gap between the fractional solutions and the dual objective value, leading to the determination of the fractional gap R_1. Subsequently, we leverage the relationship between $\widetilde{\boldsymbol{z}}(t)$ and $\overline{\boldsymbol{z}}(t)$ to forge a link between their respective server energy costs. We employ this connection as a pivotal conduit to bind the objective's remaining costs achieved by the integral solutions, leading to the determination of the rounding gap R_2. These analyses allow us to establish the competitive ratio of the ORR, denoted as $R = R_1 R_2$. Next, we will expound upon the details of our proof.

A. Feasibility Analysis and Complexity Analysis

Theorem 1: The integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$ are feasible for the problem $\mathbf{P}$.

The detailed proof is given in Appendix A, available online.

The derivation of Theorem 1 signifies that the ORR satisfies all constraints, thereby resolving the issue of necessitating repeated rounding tries, as observed in conventional randomized algorithms, to produce the feasible solutions.

Theorem 2: The ORR has polynomial time complexity.

The detailed proof is given in Appendix B, available online.

B. Equivalence Problem and Dual Problem

To facilitate our analysis, we first introduce an equivalent auxiliary problem $\mathbf{P}_1'$ for the relaxation problem $\mathbf{P}_1$ as follows:

$$\begin{aligned}\mathbf{P}_1'\text{:}\ & \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} c_n^l x_n^l(t) + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} c_n z_n(t) \\ & + \sum_{t\in\mathcal{T}}\sum_{n'\in\mathcal{N}}\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} c_{nn'}^i y_{nn'}^i(t) \\ & + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{l\in\mathcal{L}} d_n^l u_n^l(t) + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} d_n v_n(t) \qquad (3)\end{aligned}$$

s.t. (1a) - (1c)

$$\left(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} r_n^i(t) - Q_{n'}D_{n'}\right)^+ \le \sum_{n\in\mathcal{N}\setminus\{n'\}} Q_n z_n(t) \quad \forall n'\in\mathcal{N}', \forall t \qquad (3a)$$

$$u_n^l(t) \ge x_n^l(t) - x_n^l(t-1), \quad \forall l, \forall n, \forall t \qquad (3b)$$

$$v_n(t) \ge z_n(t) - z_n(t-1), \quad \forall n, \forall t \qquad (3c)$$

$$z_n(t) \ge 1, \quad \forall n, \forall t \qquad (3d)$$

$$y_{nn'}^i(t), x_n^l(t), u_n^l(t), v_n(t) \ge 0, \forall l, \forall i, \forall n, \forall n', \forall t \qquad (3e)$$

where the auxiliary variables $u_n^l(t)$ and $v_n(t)$ (associated constraint (3b), (3c) and (3e)) are equivalent to the container placement and server wake-up costs. Regarding the boxing constraint of $x_n^l(t)$, we note that there is no benefit in allowing $x_n^l(t)$ to be greater than 1. Specifically, due to the presence of constraint (1b) and the monotonicity of the objective function for $y_{nn'}^i(t)$, $y_{nn'}^i(t)$ will not be greater than $r_n^i(t)$. Similarly, due to constraint (1a) and monotonicity of the objective function for $x_n^l(t)$, we have $x_n^l(t)$ will not be greater than 1. Consequently, we can directly remove the boxing constraint of $x_n^l(t)$, as well as the boxing constraint of $u_n^l(t)$. For the boxing constraint of $z_n(t)$, we replace it with a set of knapsack cover constraints[1] [50] (i.e., constraint (3a)). Since the right-hand side of the inequality for constraint (3a) is non-negative, it is redundant when the left-hand side of the inequality is negative, that is, $\mathcal{N}' = \{n' | n' \in \mathcal{N} \cap \sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} r_n^i(t) > Q_{n'}D_{n'}\}$. Note that distinguish between $(x)^+$ and $[x]^+$, $(x)^+$ indicates a non-negative value. After that, we can remove the boxing constraint of $v_n(t)$.

We derive the dual problem $\mathbf{D}_1$ of the relaxation problem $\mathbf{P}_1$ through the Lagrangian of equivalent auxiliary problem $\mathbf{P}_1'$ as follow:

$$\begin{aligned}\mathbf{D}_1\text{:}\ & \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} \theta_n^i(t) r_n^i(t) + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} \rho_n(t) \\ & + \sum_{t\in\mathcal{T}}\sum_{n'\in\mathcal{N}'} \beta_{n'}(t)\left(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{I}} r_n^i(t) - Q_{n'}D_{n'}\right)^+ \qquad (4)\end{aligned}$$

$$\text{s.t.}\quad c_n^l - \sum_{n'\in\mathcal{N}}\sum_{i\in\mathcal{I}} \alpha_{n'n}^{il}(t) r_{n'}^i(t) + \psi_n^l(t) - \psi_n^l(t+1) \ge 0, \quad \forall l, \forall n, \forall t \qquad (4a)$$

$$c_{nn'}^i + \sum_{l\in\mathcal{L}} \alpha_{nn'}^{il}(t) h_i^l - \theta_n^i(t) + \varpi_{n'}(t) \ge 0, \quad \forall i, \forall n, \forall n', \forall t \qquad (4b)$$

$$\begin{aligned} & c_n - Q_n\varpi_n(t) - Q_n \sum_{n'\in\mathcal{N}'\setminus\{n\}} \beta_{n'}(t) - \rho_n(t) \\ & + \delta_n(t) - \delta_n(t+1) \ge 0, \quad \forall n\in\mathcal{N}', \forall t \\ & c_n - Q_n\varpi_n(t) - Q_n \sum_{n'\in\mathcal{N}'} \beta_{n'}(t) - \rho_n(t) \\ & + \delta_n(t) - \delta_n(t+1) \ge 0, \quad \forall n\in\mathcal{N}-\mathcal{N}', \forall t \end{aligned} \qquad (4c)$$

$$d_n^l - \psi_n^l(t) \ge 0, \quad \forall l, \forall n, \forall t \qquad (4d)$$

$$d_n - \delta_n(t) \ge 0, \quad \forall n, \forall t \qquad (4e)$$

where $\alpha_{nn'}^{il}, \theta_n^i(t), \varpi_n(t), \beta_n(t), \psi_n^l(t), \delta_n(t), \rho_n(t)$ are corresponding dual variables for the constraints (1a)–(1c) and constraints (3a)–(3d).

C. Competitive Ratio

We now employ the primal-dual analysis framework and KKT conditions to obtain the gap of the fractional solutions $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$. Subsequently, the relationship between $\widetilde{\boldsymbol{z}}(t)$ and $\overline{\boldsymbol{z}}(t)$ is employed as a pivotal conduit to derive the gap achieved by the integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$. In the following analysis, we use $\mathbf{D}_1^*$ to denote the optimal objective value for the dual problem $\mathbf{D}_1$.

Theorem 3: The total cost $\mathbf{P}(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t) | \forall t)$ achieved by the ORR is no greater than $R_1\mathbf{D}_1^*$, where $R_1 = 1 + \eta_x(1+\varepsilon)|\mathcal{N}| + \eta_z(1+\varepsilon')$. This implies that the fractional gap is R_1.

The detailed proof is given in Appendix D, available online.

Theorem 4: The total cost $\mathbb{E}[\mathbf{P}(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t) | \forall t)]$ achieved by the ORR is no greater than $R_2\mathbf{P}(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t) | \forall t)$, where $R_2 = \zeta_2(1+\zeta_1) + \zeta_3\zeta_2(1+\zeta_1) + \zeta_4\zeta_2(1+\zeta_1) + \zeta_5 + \zeta_6$. This implies that the rounding gap is R_2.

The detailed proof is given in Appendix E, available online.

Based on the gap of the fractional solutions $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t), \widetilde{\boldsymbol{z}}(t)$ and the gap of the final integral solutions $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)$, we derive the competitive ratio of the ORR as follow.

[1] This transformation results in the introduction of several constraints that are similar to those in constraint (3a). To handle all these constraints, one can add new dual variables and KKT conditions, similar to the analytic technique that we have used.

Theorem 5: The competitive ratio of the ORR is R_1R_2 (i.e., $\mathbb{E}[\mathbf{P}(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t), \overline{\boldsymbol{z}}(t)|\forall t)] \leq R_1R_2\mathbf{P}^*$). This implies that the total cost achieved by the ORR is no larger than R_1R_2 times the cost returned by the optimum offline solution in the worst case.

The detailed proof is given in Appendix F, available online.

VI. Evaluation

This section conducts a performance evaluation of the ORR. As far as our knowledge extends, there is currently no openly accessible dataset encompassing all the features mentioned above, including dynamic user requests across geographically dispersed edge clusters and user requests for various services along with their corresponding containers. To this end, we conduct simulation experiments that capture these characteristics, leveraging both simulated and real-world data. The parameter settings of the edge network and the user request are first illustrated. Then, the baseline algorithms are described. Finally, we present the results of the performance evaluation.

A. Evaluation Setup

Parameter Settings: Our evaluation considers multiple geographically dispersed edge clusters that can be interconnected. These edge clusters support 20 different types of services (i.e., 20 containers). The real container images and layers data, which were obtained by crawling DockerHub [51], have been utilized in multiple research papers [10], [20]. The average size of the layers comprising these container images is 1377 MB. The bandwidth of each edge cluster is configured between 200 Mbps and 1800 Mbps, while the request processing capability for each server is set between 10 and 90. Additionally, the maximum number of servers in the edge cluster ranges from 30 to 90. Considering data-intensive requests for services such as autopilot, the data size for requests is set from 12.5 MB to 50 MB. The communication delays among the edge clusters are set from 0.1 to 0.5 seconds, and the communication delays between the edge cluster and the container image registry are set from 0.5 to 1 seconds. Furthermore, we set ε and ε' to 0.1 to balance the trade-off between time efficiency and optimality of the algorithm.

User Requests: The simulated user request traces are generated using either a uniform distribution with an expected value of 50 requests or a Poisson distribution with λ ranging from 40 to 50 requests. To emulate the irregular and intermittent characteristics of edge requests, a probability of 0.4 is applied for sending requests to the edge cluster within each time slot. Moreover, the real user request traces used in the experiments are from Microsoft Azure workload [52].

Baseline Algorithms: We compare the proposed ORR algorithm against the following baseline algorithms:

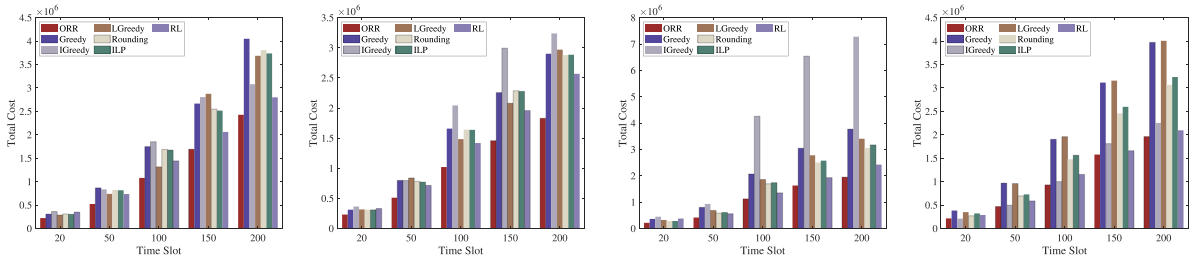
1) *Rounding* [26], which employs a linear programming technique, followed by a rounding procedure to derive the feasible solutions.
2) *IGreedy* [21], which employs iterative greedy algorithms for cluster selection to schedule requests. Subsequently, it makes decisions to enable the deployment of corresponding containers.
3) *ILP* [53], which uses the Matlab $intlinprog$ solver to solve each one-shot integer linear program separately, ignoring the switching costs.
4) *Greedy*, which solves the one-shot slice of the LJRCR at every time slot, targeting the linear costs while ignoring the switching costs.
5) *LGreedy*, which further considers the layer-sharing of container images based on the Greedy algorithm.
6) *RL* [23], which utilizes reinforcement learning to select clusters for request scheduling, followed by decision-making to enable the deployment of corresponding containers and the provisioning of necessary resources.

B. Evaluation Results

We perform experiments to evaluate the performance of the ORR in various edge computing scenarios, starting with small-scale networks featuring five geographically distributed edge clusters. The small-scale evaluation examined several critical factors, such as different types of user requests, varying time slots, various request processing capabilities, and differing bandwidths. Furthermore, to evaluate the scalability performance of the algorithm, we subsequently increase the number of edge clusters to eight. In each experiment, the parameters of the edge clusters are randomly generated, and each experiment is conducted multiple times to obtain an averaged cost measurement.

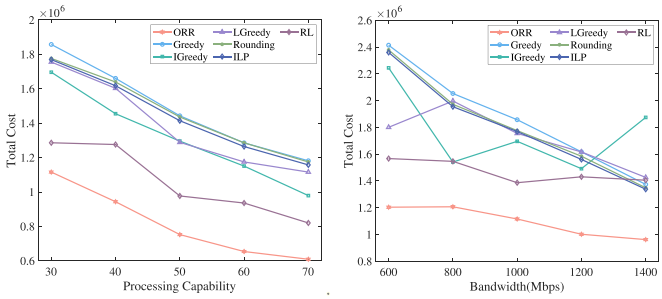
Performance with different user requests: Fig. 3 displays the total cost achieved by different algorithms for different user request traces. The ORR algorithm demonstrates superior performance compared to the baseline algorithms. In Fig. 3(a), ORR's total cost is reduced by up to 40.1% compared to the baseline algorithms. The superior performance of ORR can be primarily attributed to two key aspects. First, the formulated problem effectively integrates various cost factors involved in the container orchestration process. Second, ORR is capable of efficiently solving the formulated problem to derive appropriate decisions.

The baseline algorithms, such as Greedy, ILP, RL, etc., struggle to effectively address the challenges posed by the dynamic nature of the online problem, where future information remains unknown. These baseline methods often rely on heuristic or localized decision-making strategies, which only optimize for the immediate time slot and fail to consider the overall cost across successive time slots. For example, the ILP method, while capable of finding optimal solutions for static scenarios, suffers from computational inefficiency and is unable to adapt to the dynamic online environment. Similarly, RL approaches may require significant training data and struggle with the exploration-exploitation trade-off, leading to suboptimal performance in rapidly changing online scenarios. These baseline algorithms do not account for the interplay between multiple decisions, resulting in suboptimal strategies and higher overall costs. In contrast, the proposed ORR algorithm employs a regularization-based approach that decomposes the time-dependent switching costs into a series of time-independent terms, facilitating optimization for multiple interplay decisions. Furthermore, ORR incorporates a stepwise rounding process during the rounding of fractional



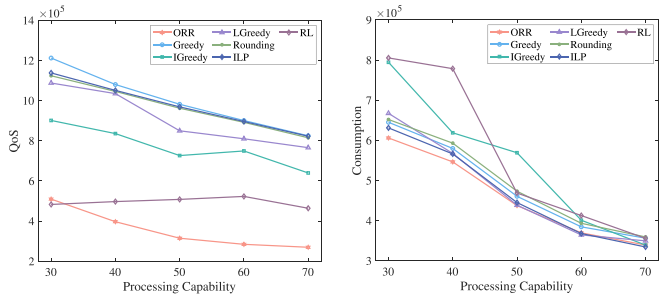
(a) Uniform distribution (b) Poisson distribution (c) Microsoft Azure workload 1 (d) Microsoft Azure workload 2

Fig. 3. The total cost incurred by different user request traces.



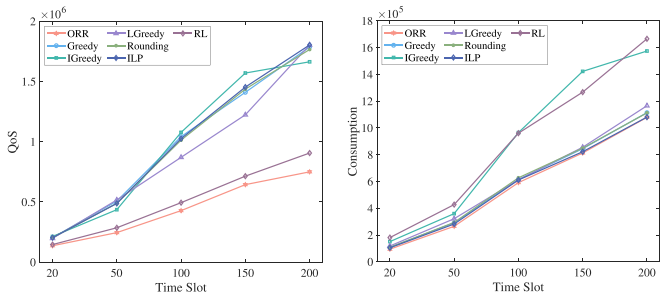
(a) Processing capability (b) Bandwidth

Fig. 4. The total cost for different request processing capabilities and bandwidths.



(a) QoS (b) Consumption

Fig. 5. The QoS and consumption for different request processing capabilities.



(a) QoS (b) Consumption

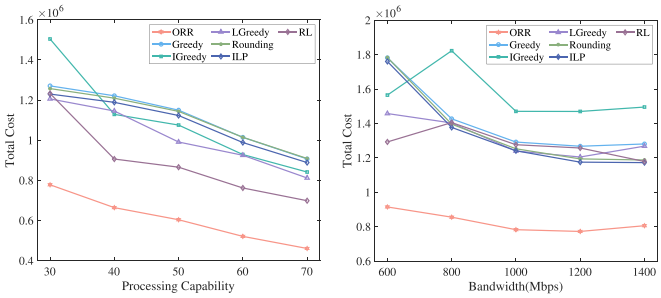
Fig. 6. The QoS and consumption for different time slots.

solutions, ensuring compensation to prevent a significant increase in total costs. By combining these two procedures, ORR achieves online decision-making with guaranteed performance, outperforming other baseline algorithms.

Impact of request processing capability and bandwidth: We next examine the impact of request processing capability (on average) and bandwidth (on average) on the total cost of several algorithms, as illustrated in Fig. 4. From Fig. 4(a), we observe that the total cost of all algorithms decreases as the request processing capacity increases. This is because a higher request processing capacity enables the edge cluster to activate fewer servers for request processing, thereby reducing server energy and server wake-up costs. Additionally, ORR outperforms baseline algorithms, reducing the total cost by at least 13.2%. This improvement is attributed to ORR's ability to make decisions that account for online dynamics and both current and past system states, whereas baseline algorithms such as Greedy, ILP, and RL primarily focus on local or heuristic-based decisions, limiting their capacity to optimize long-term performance.

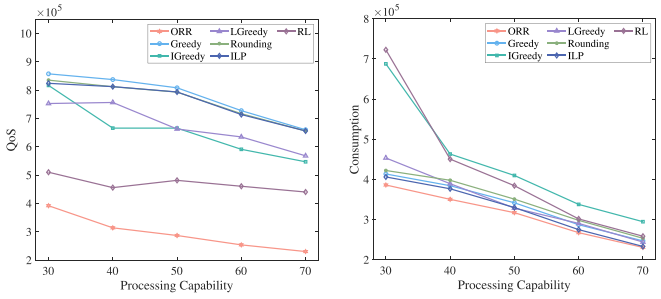
Similarly, Fig. 4(b) illustrates the general trend of decreasing total costs for all algorithms as bandwidth increases. This is because higher bandwidth accelerates the downloading of container layers from the image registry and reduces request scheduling delay. It should be noted that in each experiment, the parameters of the edge clusters were randomly generated, resulting in variants in cost due to changes in these parameters. Nonetheless, under identical conditions, the ORR algorithm consistently outperforms the baseline algorithms, achieving a reduction in total cost of at least 19.5%. The ORR algorithm leverages layer-sharing and joint decision-making to effectively handle the dynamic of online scenarios, thereby reducing the need for repeated layer downloads and exhibiting superior performance. It is worth noting that the total cost reduction trend diminishes as the bandwidth increases. This is because communication delays become the primary contributor to delay rather than bandwidth bottlenecks.

Performance of QoS and consumption: In the cost structure for the LJRCR, request scheduling, container placement, and server wake-up costs are linked to user-perceived delays, collectively termed Quality of Service (QoS). Conversely, storage and server energy costs pertain to the service provider's consumption, defined as consumption. The performance of each algorithm is assessed by decomposing the total cost into two components: QoS and consumption. To this end, Figs. 5 and 6 present the QoS and consumption for varying average request processing capabilities and time slots, respectively. As in Fig. 6(a) and (b), it can be observed that ORR's consumption is comparable



(a) Processing capability (b) Bandwidth

Fig. 7. The large-scale edge networks evaluation for different request processing capabilities and bandwidths.



(a) QoS (b) Consumption

Fig. 8. The large-scale edge networks evaluation for QoS and consumption.

to the baseline algorithms while providing significantly better QoS. This indicates that ORR can maintain relatively low consumption while offering better service to users. In addition, Fig. 5(a) illustrates that the QoS of all algorithms decreases with increased request processing capacity, mainly attributed to the reduced number of servers waken-up by the edge clusters, leading to a lower server wake-up delay. These experimental results demonstrate that the LJRCR effectively captures the multifaceted characteristics of associated container orchestrion costs, enabling the system to maintain lower consumption while providing a higher QoS to users.

Performance in large-scale edge networks: Finally, we evaluate ORR in large-scale edge computing networks by increasing the number of edge clusters to 8. Fig. 7(a) and (b) illustrate the variation in the total cost achieved by all the algorithms with different request processing capabilities (on average) and bandwidths (on average), respectively. Fig. 8(a) and (b) present the variation in QoS and consumption with different request processing capabilities (on average), respectively. The results, consistent with those obtained using 5 clusters, confirm that the proposed ORR demonstrates robust performance under these conditions.

However, as the number of edge clusters increases significantly, the proposed ORR encounters challenges related to computational efficiency. Specifically, ORR obtains the fractional solution by employing the interior-point method, which is known for its large time and space complexity, particularly as the problem size escalates. With the significant growth in the number of edge clusters, there is a corresponding increase in the number of constraints and decision variables, leading to a substantial rise in both the time required to solve the optimization problem and the memory needed to store and process the associated data structures. In extreme cases, this can exceed the available resources, resulting in premature termination due to memory limitations. This highlights a critical challenge in scaling the proposed method to accommodate a significantly larger number of edge clusters. While the ORR performs well with a moderate number of clusters, further research is required to extend the ORR that can effectively handle the increased complexity.

VII. Conclusion and Future Work

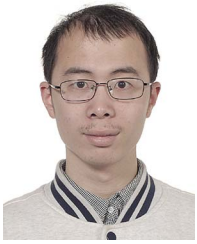
In this paper, we considered the layer-aware joint decisions of request scheduling, container placement, and resource provision in edge computing. We formulated a joint online optimization problem and proposed the ORR algorithm, which employs an online regularization-based approach combined with a stepwise rounding process as an effective solution to solve the problem. Furthermore, we provided rigorous proof of the parametric competitive ratio of the ORR. Our extensive simulation experiments validated the effectiveness of the ORR, as it achieved at about 20% improvements in performance compared to the baseline algorithms. By reducing the need for repeated layer downloads and utilizing joint decision-making strategies, ORR demonstrated superior performance in various heterogeneous scenarios. Although the ORR algorithm performed effectively with a moderate number of clusters, it faced limitations when the number of edge clusters increases significantly. Future work will extend the proposed ORR to effectively accommodate a significantly larger number of edge clusters. Additionally, implementing the proposed algorithm within a prototype system for edge computing networks based on Kubernetes would be a valuable next step.

References

[1] W. Z. Khan et al., "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, 2019.
[2] M. Billinghurst et al., "A survey of augmented reality," *Found. Trends Hum.–Comput. Interaction*, vol. 8, no. 2/3, pp. 73–272, 2015.
[3] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1243–1274, Second Quarter, 2019.
[4] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.
[5] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.
[6] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
[7] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Comput. Sci.*, vol. 171, pp. 1419–1428, 2020.
[8] Docker, 2018. [Online]. Available: https://www.docker.com/resources/what-container/
[9] A. Anwar et al., "Improving docker registry design based on production workload analysis," in *Proc. 16th USENIX Conf. File Storage Technol.*, 2018, pp. 265–278.

[10] Z. Tang, J. Lou, and W. Jia, "Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3444–3459, Jun. 2023.
[11] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *Proc. 2018 IEEE Conf. Comput. Commun.*, 2018, pp. 774–782.
[12] U. Pongsakorn, Y. Watashiba, K. Ichikawa, S. Date, and H. Iida, "Container rebalancing: Towards proactive linux containers placement optimization in a data center," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf.*, 2017, pp. 788–795.
[13] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, "DRAPS: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster," in *Proc. IEEE 36th Int. Perform. Comput. Commun. Conf.*, 2017, pp. 1–8.
[14] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "SGX-aware container orchestration for heterogeneous clusters," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 730–741.
[15] P. Kayal and J. Liebeherr, "Distributed service placement in fog computing: An iterative combinatorial auction approach," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 2145–2156.
[16] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. 2018 IEEE Conf. Comput. Commun.*, 2018, pp. 468–476.
[17] E. Oakes et al., "SOCK: Rapid task provisioning with serverless-optimized containers," in *Proc. 2018 USENIX Annu. Tech. Conf.*, 2018, pp. 57–70.
[18] Y. Li, B. An, J. Ma, and D. Cao, "Comparison between chunk-based and layer-based container image storage approaches: An empirical study," in *Proc. 2019 IEEE Int. Conf. Service-Oriented Syst. Eng.*, 2019, pp. 197–1975.
[19] 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 Trans. Netw.*, vol. 32, no. 2, pp. 1807–1822, Apr. 2024.
[20] J. Lou, H. Luo, Z. Tang, W. Jia, and W. Zhao, "Efficient container assignment and layer sequencing in edge computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1118–1131, Mar./Apr. 2023.
[21] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, "Layer aware microservice placement and request scheduling at the edge," in *Proc. 2021 IEEE Conf. Comput. Commun.*, 2021, pp. 1–9.
[22] L. Gu, D. Zeng, J. Hu, H. Jin, S. Guo, and A. Y. Zomaya, "Exploring layered container structure for cost efficient microservice deployment," in *Proc. 2021 IEEE Conf. Comput. Commun.*, 2021, pp. 1–9.
[23] X. Tian, H. Meng, Y. Shen, J. Zhang, Y. Chen, and Y. Li, "Dynamic microservice deployment and offloading for things-edge-cloud computing," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 19537–19548, Jun. 2024.
[24] S. Albers, "On energy conservation in data centers," *ACM Trans. Parallel Comput.*, vol. 6, no. 3, pp. 1–26, 2019.
[25] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[26] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, "Layer-aware collaborative microservice deployment toward maximal edge throughput," in *Proc. 2022 IEEE Conf. Comput. Commun.*, 2022, pp. 71–79.
[27] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. 2017 IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[28] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–35, 2020.
[29] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Trans. Serv. Comput.*, vol. 15, no. 5, pp. 2671–2684, Sep./Oct. 2022.
[30] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 98–115, Jan. 2021.
[31] J. Zhang, X. Zhou, T. Ge, X. Wang, and T. Hwang, "Joint task scheduling and containerizing for efficient edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 2086–2100, Aug. 2021.
[32] T. Menouer, "KCSS: Kubernetes container scheduling strategy," *J. Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.
[33] S. Li et al., "Commutativity-guaranteed docker image reconstruction towards effective layer sharing," in *Proc. ACM Web Conf.*, 2022, pp. 3358–3366.
[34] Y. Liu, B. Yang, Y. Wu, C. Chen, and X. Guan, "How to share: Balancing layer and chain sharing in industrial microservice deployment," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 2685–2698, Jul./Aug. 2023.
[35] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1751–1767, Aug. 2018.
[36] W. Chu, P. Yu, Z. Yu, J. C. S. Lui, and Y. Lin, "Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4150–4167, Jul. 2023.
[37] J. Zhang et al., "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4283–4294, Jun. 2019.
[38] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.
[39] Z. Yu, Y. Gong, S. Gong, and Y. Guo, "Joint task offloading and resource allocation in UAV-enabled mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3147–3159, Apr. 2020.
[40] F. Khoramnejad and M. Erol-Kantarci, "On joint offloading and resource allocation: A double deep Q-network approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 4, pp. 1126–1141, Dec. 2021.
[41] Z. Nan, S. Zhou, Y. Jia, and Z. Niu, "Joint task offloading and resource allocation for vehicular edge computing with result feedback delay," *IEEE Trans. Wireless Commun.*, vol. 22, no. 10, pp. 6547–6561, Oct. 2023.
[42] J. Shu, B. Li, and W. Zheng, "Design and implementation of an SAN system based on the fiber channel protocol," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 439–448, Apr. 2005.
[43] X. Shang, Y. Huang, Y. Mao, Z. Liu, and Y. Yang, "Enabling QoE support for interactive applications over mobile edge with high user mobility," in *Proc. 2022 IEEE Conf. Comput. Commun.*, 2022, pp. 1289–1298.
[44] T. Carnes and D. Shmoys, "Primal-dual schema for capacitated covering problems," in *Proc. Int. Conf. Integer Program. Combinatorial Optim.*, Springer, 2008, pp. 288–302.
[45] K. Poularakis et al., "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1047–1060, Jun. 2020.
[46] N. Buchbinder, S. Chen, and J. Naor, "Competitive analysis via regularization," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, SIAM, 2014, pp. 436–444.
[47] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia, PA, USA: SIAM, 1994.
[48] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, "Dependent rounding and its applications to approximation algorithms," *J. ACM*, vol. 53, no. 3, pp. 324–360, 2006.
[49] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[50] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips, "Strengthening integrality gaps for capacitated network design and covering problems," in *Proc. 11th Annu. ACM-SIAM Symp. Discrete Algorithms*, SIAM, 2000, pp. 106–115.
[51] Dockerhub, 2014. [Online]. Available: https://hub.docker.com
[52] M. Shahrad et al., "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. 2020 USENIX Annu. Tech. Conf.*, 2020, pp. 205–218.
[53] intlinprog solver, 2014. [Online]. Available: https://www.mathworks.com/help/optim/ug/intlinprog.html

Zhenzheng Li received the BS degree from the School of Information Technology, Beijing Institute of Technology, Zhuhai, China, in 2019, and the MS degree from the School of Information and Control Engineering, China University of Mining and Technology, China, in 2022. He is currently working toward the PhD degree with the School of Artificial Intelligence, Beijing Normal University, China. His current research interests include edge computing and resource scheduling.



Jiong Lou (Member, IEEE) received the BS and PhD degrees from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2016 and 2023, respectively. Since 2023, he has held the position of research assistant professor with 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., *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing* and *IEEE Transactions on Services Computing*). His current research interests include edge computing, task scheduling and container management. He has served as a reviewer for *Computer Networks*, *Journal of Parallel and Distributed Computing*, *IEEE Internet of Things Journal*, and ICDCS.

Zhiqing Tang (Member, IEEE) received the BS degree from the School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015, and the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2022. He is currently an assistant professor with the Advanced Institute of Natural Sciences, Beijing Normal University, China. His research interests include edge computing, resource scheduling, and reinforcement learning.

Jianxiong Guo (Member, IEEE) received the BE degree from the School of Chemistry and Chemical Engineering, South China University of Technology, Guangzhou, China, in 2015, and the PhD degree from the Department of Computer Science, University of Texas at Dallas, Richardson, Texas, in 2021. He is currently an associate professor with the Advanced Institute of Natural Sciences, Beijing Normal University, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai, China. He is a member of ACM/CCF. His research interests include social networks, wireless sensor networks, combinatorial optimization, and machine learning.

Tian Wang (Senior Member, IEEE) received the BSc and MSc degrees in computer science from Central South University, in 2004 and 2007, respectively, and the PhD degree in computer science from the City University of Hong Kong, in 2011. Currently, he is a professor with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University. His research interests include the Internet of Things, edge computing, and mobile computing. He has 27 patents and has published more than 200 papers in high-level journals and conferences. He has more than 14000 citations, according to Google Scholar. His H-index is 68. He has managed six national natural science projects (including 2 sub-projects) and four provincial-level projects.

Weijia Jia (Fellow, IEEE) is currently the director of Institute of Artificial Intelligence and Future Networking, and the director of Super Intelligent Computer Center, Beijing Normal University, Zhuhai, also a chair professor with UIC, Zhuhai, Guangdong, China. He has served as the VP for research with UIC, in 6/2020-7/2024. Prior joining BNU, he served as the deputy director of State Key Laboratory of Internet of Things for Smart City, The University of Macau and Zhiyuan chair professor with Shanghai Jiaotong University, PR China. From 95-13, he worked with the City University of Hong Kong as a professor. His contributions have been recoganized for the research of edge AI, optimal network routing and deployment, vertex cover, anycast and multicast protocols, sensors networking, knowledge relation extractions, NLP and intelligent edge computing. He has more than 700 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), and top two percent World Scientists in Stanford-list (2020–2024) and many provincial science and tech awards. He has served as area editor for various prestige international journals, chair and PC member/keynote speaker for many top international conferences. He is the distinguished member of CCF.

Wei Zhao (Fellow, IEEE) received the undergraduate degree in physics with Shaanxi Normal University, China, in 1977, and the MSc and PhD degrees in computer and information sciences from the University of Massachusetts at Amherst, in 1983 and 1986, respectively. He has served important leadership roles in academic including the chief research officer with the American University of Sharjah, the chair of Academic Council with CAS Shenzhen Institute of Advanced Technology, the eighth Rector of the University of Macau, the dean of science with 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 with Texas A&M University. He 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. He was awarded the Lifelong Achievement Award by the Chinese Association of Science and Technology, in 2005.