Toward Online Reliability-Enhanced Microservice Deployment With Layer Sharing in Edge Computing

You Shi, Yuye Yang, Changyan Yi, *Member, IEEE*, Bing Chen, and Jun Cai, *Senior Member, IEEE*

***Abstract*—Container–based microservice provisioning, with its elasticity in terms of the layered structure, enables the sharing of common layers among different edge computing tasks, both within and across edge servers (ESs). However, due to the potential hardware breakdowns, each ES may prone to failures, affecting its lifetime (i.e., the time-length that an ES works continuously without interruptions), and in turn leading to the collapse of their hosted/provided microservices or the other ESs' microservices requesting common layers from it. To address such an issue, in this article, we study the microservice deployment optimization with layer sharing for maximizing the system-wide reliability while satisfying all tasks' delay requirements. Considering dynamic task generations and the asynchronization of various decision variables with different triggers, we design an online optimization algorithm by leveraging an improved Lyapunov technique integrating randomized rounding, Lagrangian method and convex optimization, which iteratively solves the problem over different timescales. Theoretical analyses and simulations evaluate the performance of the proposed solution, showing that it can achieve an increase of 12.4% in reliability and a reduction of 28.57% in total delay, compared to the counterparts.**

***Index Terms*—Edge computing, layer sharing, microservice deployment, online optimization, reliability enhancement.**

I. Introduction

EDGE computing has been proposed as a compelling alternative or supplement to cloud computing by providing services at the vicinity of mobile users [1], [2]. To cope with the sky-rocketing amount of computing services and extremely high-quality while low-latency user demands, it is desirable to have light-weight and easydeploying service provision at the edge servers (ESs). This necessitates an emerging technique, called container-based microservice provisioning [3], [4]. Via this approach, multiple containers can share the machine's operating system (OS) kernel and thereby do not require an OS per container, driving higher efficiencies, suitable for resource-limited edge nodes. Microservice deployment optimization is one of the main issues in microservice orchestration, and has been widely studied toward various objectives, e.g., resource utilization maximization [5], [6], system cost minimization [7], [8] and startup time acceleration [9].

Each container-based microservice is in a layered structure, which packages all required items, such as runtime tools, system tools, libs, and system dependencies, in different layers [9], while different container-based microservices may share several common base layers. For example, Cassandra, JAVA and Python are all based on the same nonlatest Linux distribution layer of Debian [10]. By such a way, at a minimum, only one copy of the shared common layer need to be downloaded from the cloud repository to ESs. In addition, recent studies have revealed that common layers of microservices can be shared by computing tasks executed on the same ES (i.e., intra-ES sharing) [9] or across different ESs via distributed file systems (i.e., inter-ESs sharing) [11]. Specifically, for each ES, on the one hand, the common layer only needs to place one copy that can be shared among the tasks executing on the same ES. On the other hand, if an ES lacks sufficient cache capacity to host certain common layers locally, it can load the required common layers from other ESs that have already deployed them using distributed file system. All these indicate that the microservice deployment problem is essentially equivalent to the layer placement and loading problems, i.e., which layer should be placed directly on each ES, and which layer should be loaded by each required ES from which ESs having already placed this layer.

It is worth noting that each ES may suffer from a variety of runtime failures caused by potential hardware breakdowns and configuration errors [12], [13], [14]. This can significantly affect ESs' lifetime (i.e., the time-length that an ES works continuously without interruptions), and consequently reducing the reliability of their hosted/provided container-based microservices in serving computing tasks [13]. To be more specific, when each task arrives at its associated ES requesting a certain microservice, it may be scheduled to any ES (including its associated one) depending on the resource capacities and layer placement and loading decisions. If an ES crashes afterwards, not only its microservices for the assigned

Manuscript received 7 February 2024; accepted 27 March 2024. Date of publication 8 April 2024; date of current version 26 June 2024. This work was supported in part by the State Key Laboratory of Massive Personalized Customization System and Technology under Grant H&C-MPC-2023-04-01; in part by the National Natural Science Foundation of China (NSFC) under Grant 62176122; in part by the A3 Foresight Program of NSFC under Grant 62061146002; and in part by the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant KYCX22_0372. *(Corresponding authors: Changyan Yi; Bing Chen.)*

You Shi, Yuye Yang, Changyan Yi, and Bing Chen are with the College of Computer Science and Technology and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, Jiangsu, China (e-mail: shyou@nuaa.edu.cn; mryyy@nuaa.edu.cn; changyan.yi@nuaa.edu.cn; cb_china@nuaa.edu.cn).

Jun Cai is with the Department of Electrical and Computer Engineering, Concordia University, Montréal, QC, H3G 1M8, Canada (e-mail: jun.cai@concordia.ca).

Digital Object Identifier 10.1109/JIOT.2024.3385816

tasks collapse, but also the other ESs' mircoservcies loading common layers from it for serving all other tasks collapse.

Enhancing the system-wide reliability by utilizing the aforementioned features is of great importance, while this has received limited attention in the existing literature. Technically, such an issue is challenged by the following aspects.

1) To maximize the system-wide reliability of edge computing, it is necessary to jointly optimize the task scheduling among ESs, layer placement and loading of container-based microservices, along with the computing resource allocation. Furthermore, since ESs may encounter failures accidentally, meaning that their lifetime is uncertain, and tasks may be generated randomly requesting heterogeneous microservices, all optimization decisions should be dynamically adjusted. This results in an online optimization and its long-term performance guarantee requires the statistics of future network dynamics, which is difficult, if not impossible, to be obtained [15].
2) For each ES, frequent layer placements may cause the storage space fragmentation [16], while the layer loading, task scheduling and computing resource allocation are triggered by task generations and ESs' runtime failures, which need to be adapted in a much higher frequency. These imply that decision variables in such an online problem should be optimized asynchronously in different timescales rather than a single one as those in conventional studies [9], [17].

To fill the gap of the literature, in this article, we design a novel two-timescale online management framework for edge computing with container-based microservice provisioning, including the optimization of 1) placing which layer on each ES in the large timescale and 2) selecting which ES to load which required layer for each ES and determining the computing resource allocation and task scheduling in the small timescale. Particularly, we aim to maximize the long-term average reliability of all microservices in serving computing tasks while ensuring not only the system stability but also the constrained delay requirements and caching capacities. Besides, the uncertainty of ESs' lifetime and the randomness of task generations requesting different microservices are taken into account. To this end, we propose an online algorithm based on the Lyapunov optimization method but has an improved structure for dealing with two-timescale decision variables. On top of this, by further decoupling decisions into different timescales, we develop an iterative algorithm iterating randomized rounding, Lagrangian method and convex optimization. Theoretical analyses show that the proposed solution can well address the original problem to the asymptotic optimum with a low-computational complexity.

The main contributions of this article are summarized in the following.

1) To the best of our knowledge, this is the first work studying the reliability enhancement of container-based microservices in edge computing with layer sharing. Specifically, we formulate a two-timescale online optimization problem for adaptively determining the task scheduling among ESs, the layer placement and loading of different microservices along with the computing resource allocation under network dynamics, for maximizing the long-term system-wide average reliability.
2) Taking an equivalent problem reformulation, we propose a low-complexity online algorithm based on the improved Lyapunov optimization method, which decomposes the long-term problem into a series of deterministic ones. Then, we further decouple decisions into two different timescales, and develop an iterative algorithm to reach a near-optimum.
3) Extensive theoretical analyses and numerical simulations are conducted to examine the feasibility of the proposed solution, and demonstrate its superiority compared to the counterparts.

The remainder of this article is organized as follows. We discuss related works in Section II. In Section III, we introduce the system model with the definition of system-wide reliability. Section IV shows the considered problem formulation. Section V presents the detail of the proposed solution along with comprehensive analyses. Simulation results are given in Section VI, followed by conclusions in Section VII.

II. Related Work

Container-based microservice provisioning has emerged as a promising way in promoting edge computing elasticity. Recently, container-based microservice deployment has been widely studied targeting for different objectives. For instance, Guo and Yao [18] introduced a novel container-based microservice scheduling strategy to provide faster microservice response by considering edge load balancing and response time. Lv et al. [19] proposed a graph-reinforcement learning-based microservice deployment framework, which utilized graph convolutional networks to deal with differentiated container dependencies and scheduling conflicts in edge computing. Shi et al. [20] investigated a deep learning algorithm for automatic scaling of container-based microservice deployments to fit high-state spaces in distributed cloud environments. In [21], the starting time of container-based microservices and the interference of resource contention among multiple microservices were jointly considered, and a low-complexity heuristic algorithm was proposed to minimize the number of servers, thereby reducing the system overhead. However, most of the existing researches on container-based microservice deployment treats the containers as a whole, and rarely considers the hierarchical structure of container images.

Considering the intrinsic layered structure of containers, layer placement and loading are envisioned as a promising way to provide flexible microservice deployment. Zhou et al. [17] formulated the layer placement of microservices requested by cellular users as an optimization problem, aiming for minimizing the energy cost. Tang et al. [22] proposed a novel online container migration algorithm for reducing task delay in edge-assisted vehicular networks by leveraging multiuser layer-sharing information. Gu et al. [9], [23] proposed an iterative greedy-based intra-ES layer sharing microservice deployment solution to improve the edge

TABLE I
TABLE COMPARING OUR WORK WITH THE EXISTING STUDIES

Reference	Objective	Method	Reliability	Layer placement	Layer loading
[7]	Network throughput maximization	Randomized rounding approach	✗	✓	✓
[18]	Optimization of load balancing	Neighborhood division-based container scheduling strategy	✗	✗	✓
[19]	Request response time minimization	Graph reinforcement learning approach	✗	✓	✗
[20]	Optimization of container scaling	Reinforcement learning approach	✗	✓	✗
[21]	Network utility maximization	Low-complexity heuristic approach	✗	✓	✗
[17]	Optimization of average energy consumption	Particle swarm-based optimization approach	✗	✓	✗
[22]	Container migration delay minimization	Gradient-based reinforcement learning algorithm	✗	✓	✗
[23]	Microservice deployment cost minimization	Accelerated distributed augmented Lagrangian algorithm	✗	✓	✗
[24]	Instant service reliability maximization	MAB-based approximation method	✓	✓	✗
[12]	Equilibrium between reliability and bandwidth cost	Successive convex approximation	✓	✗	✗
[14]	System lifetime maximization	Hybrid offline and online algorithm	✓	✗	✗
Proposed work	System-wide reliability maximization	Two-timescale online optimization approach	✓	✓	✓

throughput. Gu et al. [7] applied a linear programming method to maximize the service capacity of ESs by enabling both intra-ES and inter-ES layer sharing. However, all these papers assumed that ESs were able to work permanently without facing any breakdowns or errors.

Some recent works have been dedicated in studying the impacts of ESs' failures on their service reliability. For example, Ma et al. [24] proposed a reliability-aware scheme for edge computing by leveraging both online feedback and offline data to maximize both the accuracy and reliability of DNN inference tasks. Liu et al. [12] investigated the equilibrium of minimizing the bandwidth consumption of IoT applications while maximizing the reliability of IoT users with uncertain lifetime of ESs. Cao et al. [14] jointly optimized the execution delay and lifetime of an edge-intelligent system subject to the constraints of reliability and energy. However, all of them did not focus on the container-based microservice provisioning, ignoring the potential reliability issues caused by layer sharing.

In summary, this work differs from the existing literature (as demonstrated in Table I) by the following aspects: 1) different from the existing works that only consider the service reliability of microservices deployed within a single ES, this article quantifies the service reliability of the whole system of edge computing by jointly considering the microservice deployment under intra-ES sharing and inter-ESs sharing; 2) a novel reliability-enhanced dynamic microservice deployment framework is investigated, where key issues, such as layer placement and loading, task scheduling, and computing resource allocation across multiple ESs, are jointly optimized; and 3) aiming at addressing the asynchronization of the decision variables, a two-timescale online algorithm based on the modified Lyapunov technique is proposed achieving the near-optimum.

III. SYSTEM MODEL

In this section, the system model of microservice deployment decision, layer placement, layer loading and task scheduling for resource-constrained edge computing is described. For convenience, Table II lists some important notations used in this article.

TABLE II
IMPORTANT NOTATIONS IN THIS ARTICLE

Symbol	Meaning
$\mathcal{M}$	set of ESs
$\mathcal{S}$	set of all kinds of microservices
v_s	unit task size of microservice s
v_l	the size of layer l
$\lambda_m^s(t)$	microservice manu variable of ES m in time frame t
$r_{m',m}^{tra}(\tau)$	transmission rate between ES m' and m in time frame τ
$r_{m,m'}^{loa}(t)$	loading rate between ES m and m' in time frame t
$x_{m,m'}^s(\tau)$	task scheduling variable for microservice s in time slot τ
$\vartheta_{m,m'}^l(\tau)$	layer loading variable for ES m in time slot τ
$d_m^l(t)$	layer placement variable for ES m in time frame t
$\rho_m^s(\tau)$	computing resource allocation for ES m in time slot τ
$D_{m',m}^{s,tra}(\tau)$	transmission delay for microservice s's task scheduling
$D_{m,s}^{com}(\tau)$	computing delay for microservice s's task processing
$D_m^{pla}(t)$	layer placement delay for ES m in time frame t
$D_m^{loa}(\tau)$	layer loading delay for ES m in time slot τ
$D_m^{life}(\tau)$	the life time of ES m in time slot τ
$D_{m,s}^{upt}(\tau)$	the uptime of microservice s on ES m in time slot τ
V	Lyapunov control parameter
$REL_m^s(\tau)$	reliability in serving microservice s's tasks in time slot τ

A. Overview of the System

Consider an edge computing system, as depicted in Fig. 1, consisting of multiple geographically distributed ESs, represented by set $\mathcal{M}$ with a cardinality of $|\mathcal{M}| = M$, and a container repository, which is deployed on a remotely located cloud center, that stores a set $\mathcal{L}$ of container layers for supporting different microservices. Each ES can host several microservices, and the set of all kinds of microservices can be denoted by $\mathcal{S} = \{1, 2, \ldots, S\}$, where each of them aims to handle a specific type of tasks. If an ES intends to serve a certain type of tasks, it has to prepare all layers of the microservice requested by such type of tasks. This can be

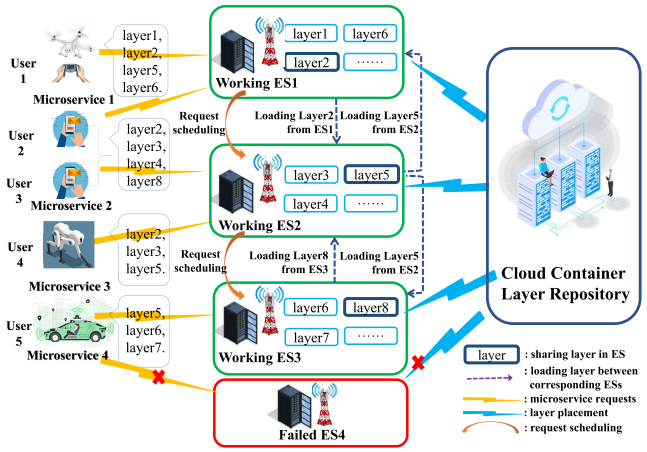

Fig. 1. Illustration of the considered system.

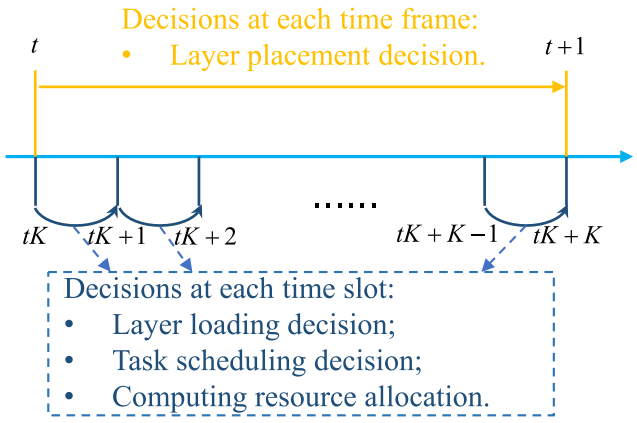

Fig. 2. Two-timescale network optimization for MEC.

done by either placing the required layers downloaded from the cloud repository or load them from other peer ESs having already placed these layers. Let $\alpha_m^s(\tau) \in \mathbb{N}^+$ be the amount of tasks that arrive on ES m and request microservice $s \in \mathcal{S}$, and their unit task size (measured by bits) be v_s. Furthermore, whenever a task is generated and transmitted to the connected ES, it may be flexibly scheduled to any ES for execution, depending on the resource capacities, layer placement and loading decisions, and most importantly the risk of ESs' failures.

Note that, in practice, layer placement may not be able to vary frequently in real-time because of the storage space fragmentation [25]. In contrast, the layer loading, task scheduling and computing resource allocation require immediate and frequent responses to accommodate time-varying task generations and uncertain ESs' runtime failures. To this end, we propose an online optimization framework, where the layer placement is operated at the large timescale, while the layer loading, task scheduling and computing resource allocation are operated at the small timescale, as shown in Fig. 2. Specifically, the timeline is divided into $T \in \mathbb{N}^+$ coarse-grained time frames, and each frame can be further regarded as a combination of $K \in \mathbb{N}^+$ fine-grained time slots, where the length of each slot is γ. Define $t = \{0, 1, \ldots, T-1\}$ be the index of the tth time frame, and the index of the τth time slot in the tth time frame as $\tau \in \mathcal{T}_t = \{tK, tK+1, \ldots, tK+K-1\}$.

In summary, we aim to enhance the system-wide long-term reliability of edge computing by determining 1) which layers of microservices should be placed on each ES in each time frame and 2) which layer should be loaded by each required ES from which ESs and the corresponding computing resource allocation and task scheduling in each time slot, in an online manner.

B. Task Scheduling and Computation

Container-based microservices are often managed by applying container orchestration tool (e.g., Kubernetes [26]), which can adjust the microservice menu of ESs based on the requests of tasks, facilitating a better organization of layer placement and loading [9]. Since the preparation of such miroservice menus may be complicated and costly for ESs [27], we introduce a large-timescale variable $\lambda_m^s(t) \in \{0, 1\}$ to denote whether microservice s should be added to the microservice menu of ES m ($\lambda_m^s(t) = 1$) or not ($\lambda_m^s(t) = 0$) in each time frame t. This implies that each ES's microservice menu changes over time frames rather than time slots, resulting in a relatively small overhead. We represent the task scheduling decision as variables $x_{m,m'}^s(\tau) \in \mathbb{N}^+$ indicating the amount of tasks requesting microservice s scheduled from ES m to another ES $m' \in \mathcal{M}$. Obviously, the tasks requesting microservice s from ES m can be scheduled to ES m' if and only if m' has included s in this manu, and thus

$$x_{m,m'}^s(\tau) \le \lambda_{m'}^s(t)\alpha_m^s(\tau) \quad \forall s \in \mathcal{S}. \tag{1}$$

Here, we do not rule out the possibility that $m = m'$, so that we have

$$\sum_{m' \in \mathcal{M}} x_{m,m'}^s(\tau) = \alpha_m^s(\tau) \quad \forall s \in \mathcal{S}. \tag{2}$$

Furthermore, the transmission delay for the tasks requesting microservice $s \in \mathcal{S}$ scheduled from ES m' to m in time slot $\tau \in \mathcal{T}_t$ can be calculated as

$$D_{m',m}^{s,tra}(\tau) = \frac{x_{m',m}^s(\tau) v_s}{r_{m',m}^{\text{tra}}(\tau)} \tag{3}$$

where $r_{m',m}^{\text{tra}}(\tau)$ is the transmission rate between m' and m. Then the maximum delay for ES m to collect all tasks requesting microservice s from all ESs can be given by

$$D_{m,s}^{\text{sch}}(\tau) = \max\left\{D_{m',m}^{s,\text{tra}}(\tau)\right\} \quad \forall m' \in \mathcal{M}. \tag{4}$$

For each ES m, denote $\rho_m^s(\tau) \in [0, 1]$ as a small-timescale decision variable indicating the proportion of edge computing resource allocated to tasks requesting microservice $s \in \mathcal{S}$ in time slot $\tau \in \mathcal{T}_t$, which should meet the following conditions:

$$\sum_{s \in \mathcal{S}} \rho_m^s(\tau) \le 1 \quad \forall s \in \mathcal{S}. \tag{5}$$

Then, for each type of tasks requesting microservice s in ES m, the edge processing delay for executing all of them in time slot $\tau \in \mathcal{T}_t$ can be given by

$$D_{m,s}^{\text{com}}(\tau) = \frac{\left[\alpha_m^s(\tau) + \sum_{m' \in \mathcal{M}}\left(x_{m',m}^s(\tau) - x_{m,m'}^s(\tau)\right)\right] v_s \beta_s}{\rho_m^s(\tau) f_m^{es}} \tag{6}$$

where f_m^{es} represents CPU computation speed (measured by cycles/s) of each ES $m \in \mathcal{M}$, and β_s is the number of CPU cycles required to complete each bit of these tasks.

C. Layer Placement and Loading

Let the large-timescale decision variable $d_m^l(t) \in \{0, 1\}$ denote whether layer $l \in \mathcal{L}$ should be placed from the cloud repository to ES $m \in \mathcal{M}$ ($d_m^l(t) = 1$) or not ($d_m^l(t) = 0$). It is worth noting that the total amount of layers that can be cached on each ES $m \in \mathcal{M}$ is limited by ES m's caching capacity, denoted by Ω_m^{ca}. Therefore, we have

$$\sum_{l \in \mathcal{L}} d_m^l(t) v_l \leq \Omega_m^{ca} \quad \forall m \in \mathcal{M} \tag{7}$$

where v_l is the size (measured by bits) of layer $l \in \mathcal{L}$.

Besides, for each ES $m \in \mathcal{M}$, the layer placement delay $D_m^{\text{pla}}(t)$ can be calculated as

$$D_m^{\text{pla}}(t) = \sum_{l \in \mathcal{L}} d_m^l(t) v_l / r_{m,c}^{\text{pla}}(t) \tag{8}$$

where $r_{m,c}^{\text{pla}}(t)$ is the transmission rate from cloud to ES m.

Let the small-timescale variable $\vartheta_{m,m'}^l(\tau) \in \{0, 1\}$ be the layer loading decision of ES m in time slot τ, where $\vartheta_{m,m'}^l(\tau) = 1$ indicates that layer l will be loaded from ES $m' \in \mathcal{M}$, and $\vartheta_{m,m'}^l(\tau) = 0$ otherwise. Specifically, a layer l can be loaded from ES m' only if ES m' has placed the intended layer l, i.e.,

$$\vartheta_{m,m'}^l(\tau) \leq d_{m'}^l(t) \quad \forall l \in \mathcal{L} \ \forall m, m' \in \mathcal{M}. \tag{9}$$

Note that there may coexist multiple replicas of the same layer placed on different ESs. Each ES $m \in \mathcal{M}$ can select at most one ES $m' \in \mathcal{M}$ (including itself, i.e., $m = m'$) to load a certain layer, i.e.,

$$\sum_{m' \in \mathcal{M}} \vartheta_{m,m'}^l(\tau) \leq 1 \quad \forall l \in \mathcal{L} \quad \forall m, m' \in \mathcal{M}. \tag{10}$$

For each ES $m \in \mathcal{M}$, the corresponding layer loading delay is introduced in all time slots, calculated as the maximum delay of loading all required layers from all ESs, i.e.,

$$D_m^{\text{loa}}(\tau) = \max\left\{ \frac{\vartheta_{m,m'}^l(\tau) v_l}{r_{m,m'}^{\text{loa}}(t)} \right\} \quad \forall l \in \mathcal{L} \quad \forall m' \in \mathcal{M} \tag{11}$$

where $r_{m,m'}^{\text{loa}}(t)$ denotes the loading rate between ES m and ES m' on a wired communication link (which is assumed to be invariant within t [28], [29]).

For each container-based microservice $s \in \mathcal{S}$ provided by each ES m, it can be successfully initiated only if all required layers are completely prepared via layer placement and loading. That is

$$\lambda_m^s(t) \varpi_s^l \leq d_m^l(t) + \sum_{m' \in \mathcal{M}} \vartheta_{m,m'}^l(\tau) \tag{12}$$

where binary variable $\varpi_s^l \in \{0, 1\}$ indicates whether microservice $s \in \mathcal{S}$ requires layer $l \in \mathcal{L}$ ($\varpi_s^l = 1$) or not ($\varpi_s^l = 0$).

Taking into account all above, the total delay for tasks requesting microservice $s \in \mathcal{S}$ at ES m in each time frame t can be derived as

$$D_{m,s}^{\text{tol}}(t) = D_m^{\text{pla}}(t) + \sum_{\tau \in \mathcal{T}_t} D_{m,s}^{\text{exe}}(\tau) \tag{13}$$

where $D_m^{\text{pla}}(t)$ is the layer placement delay in time frame t, and $D_{m,s}^{\text{exe}}(\tau)$ is the edge execution delay for executing tasks requesting microservice $s \in \mathcal{S}$ in time slot $\tau \in \mathcal{T}_t$, i.e.,

$$D_{m,s}^{\text{exe}}(\tau) = D_m^{\text{loa}}(\tau) + D_{m,s}^{\text{com}}(\tau) + D_{m,s}^{\text{sch}}(\tau) \tag{14}$$

consisting of layer loading delay $D_m^{\text{loa}}(\tau)$, edge processing delay $D_{m,s}^{\text{com}}(\tau)$ and the delay to collect all tasks requesting miroservice s, i.e., $D_{m,s}^{\text{sch}}(\tau)$.

IV. Problem Formulation

The lifetime of an ES refers to the time-length from the point of starting the operation to that of occurring malfunctions [14]. Following the conventions in [12] and [30], in each time slot $\tau \in \mathcal{T}_t$, the lifetime $D_m^{\text{life}}(\tau)$ of ES $m \in \mathcal{M}$ is defined to be an exponentially distributed random variable with parameter $c_m(\tau) \in [c_m^{\min}, c_m^{\max}]$, meaning that

$$\Pr\left(D_m^{\text{life}}(\tau) \leq \gamma\right) = 1 - e^{-c_m(\tau)\gamma} \tag{15}$$

where γ is the length of each time slot.

Due to the layer sharing, lifetime of an ES affects not only its own hosted microservices for the assigned tasks, but also the other ESs' microservices loading common layers from it. In other words, if a certain ES collapses, it can potentially disrupt the availability and the normal operation time-length (i.e., uptime) of all microservices supported by any of its shared layers. Additionally, since the lifetime of different ESs are independent, the uptime of a microservice on each ES's menu actually corresponds to the minimum lifetime of all ESs that provide common layers to this ES (i.e., all ESs loaded by this ES for acquiring common layers). Accordingly, the uptime of microservice $s \in \mathcal{S}$ on ES $m \in \mathcal{M}$ in time slot τ can be expressed as

$$D_{m,s}^{\text{upt}}(\tau) = \min\{D_{m'}^{\text{life}}(\tau)\}, m' \in \mathcal{M}'(\tau) \tag{16}$$

where $\mathcal{M}'(\tau) = \{m' | \vartheta_{m,m'}^l(\tau) = 1, m' \in \mathcal{M}, l \in \mathcal{L}\}$ represents the set of ESs loaded by ES m for acquiring common layers in serving tasks requesting microservice s in time slot τ. Note that, if the edge execution delay $D_{m,s}^{\text{exe}}(\tau)$ exceeds $D_{m,s}^{\text{upt}}(\tau)$, the microservice for this type of tasks will collapse. Consequently, the probability that microservice s on ES m collapses in time slot τ, denoted by $F_{m,s}^{\text{fail}}(D_{m,s}^{\text{exe}}(\tau))$, can be calculated as

$$\begin{aligned} F_{m,s}^{\text{fail}}\left(D_{m,s}^{\text{exe}}(\tau)\right) &= Pr\left\{D_{m,s}^{\text{upt}}(\tau) \leq D_{m,s}^{\text{exe}}(\tau)\right\} \\ &= 1 - Pr\left\{D_{m,s}^{\text{upt}}(\tau) > D_{m,s}^{\text{exe}}(\tau)\right\} \\ &= 1 - \prod_{m' \in \mathcal{M}'(\tau)} Pr\left\{D_{m'}^{\text{life}}(\tau) > D_{m,s}^{\text{exe}}(\tau)\right\} \\ &= 1 - e^{-\sum_{m' \in \mathcal{M}'(\tau)} c_{m'}(\tau) D_{m,s}^{\text{exe}}(\tau)}. \end{aligned} \tag{17}$$

Then, the reliability in serving tasks requesting microservice s at ES m in time slot $\tau \in \mathcal{T}_t$ can be computed by

$$\text{REL}_m^s(\tau) = 1 - F_{m,s}^{\text{fail}}\left(D_{m,s}^{\text{exe}}(\tau)\right) \quad \forall s \in \mathcal{S} \quad \forall m \in \mathcal{M}. \tag{18}$$

Taking the average reliability of all microservices over all time frames as the performance measurement, the system-wide reliability of edge computing can be defined as

$$\mathcal{R} = \frac{1}{TK} \sum_{\tau=0}^{KT-1} \sum_{m\in\mathcal{M}} \sum_{s\in\mathcal{S}} \mathrm{REL}_m^s(\tau). \tag{19}$$

For maximizing $\mathcal{R}$, we are required to optimize 1) microservice manu decision in any time frame t; 2) layer placement decision for each ES m in any time frame t; 3) layer loading decision for each ES m in any time slot $\tau \in \mathcal{T}_t$; 4) task scheduling decision in any time slot $\tau \in \mathcal{T}_t$; and 5) the computing resource allocation in any time slot $\tau \in \mathcal{T}_t$, denoted in short by $\mathcal{J}_A(t) = \{\lambda_m^s(t), d_m^l(t)\}$, $\mathcal{J}_B(\tau) = \{\vartheta_{m,m'}^l(\tau), \rho_m^s(\tau), x_{m,m'}^s(\tau)\}$ $\forall s \in \mathcal{S}$ $\forall m \in \mathcal{M}$. Mathematically, such a two-timescale long-term reliability maximization problem can be formulated as

$$\begin{aligned} \mathcal{P}_0: \max_{\mathcal{J}_A(t),\mathcal{J}_B(\tau)} \ & \lim_{T\to\infty} \mathcal{R} \\ \text{s.t.} \ & (1),\ (2),\ (5),\ (7)-(12) \\ & \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} D_{m,s}^{\mathrm{tol}}(t) \le D_{m,s}^{\mathrm{th}} \end{aligned} \tag{20}$$

where besides the aforementioned (1), (2), (5), (7)–(12), and (20) is the long-term average delay constraint, in which $D_{m,s}$th denotes a predetermined delay threshold of all tasks requesting microservice s.

Remark 1: It is obvious that problem $\mathcal{P}_0$ is a two-timescale stochastic optimization problem, in which the microservice menu decision and layer placement are operated at the large timescale, and the layer loading, task scheduling and computing resource allocation are operated at the small timescale, with the objective of maximizing the long-term reliability. Solving problem $\mathcal{P}_0$ is very challenging because of the following reasons.

1) With the rapid proliferation of MDs and microservice requests, the vast action and state space pose significant computational complexity, as well as uncertain data noise, making the amount of historical information extremely large. Consequently, the reinforcement learning methods [31] that rely on large-scale trial and error and extensive training to discover optimal solutions face potential issues, such as slow convergence speed and substantial fluctuations in the obtained solutions.
2) Furthermore, (2) and (5) indicate the inclusion of discrete decision variables, while (7)–(12) are nonlinear, and moreover $\mathcal{J}_A(t)$ and $\mathcal{J}_B(\tau)$ are operated at different timescales, making $\mathcal{P}_0$ become a two-timescale mixed integer nonlinear programming problem (two-timescale MINLP).

This indicates that the traditional Lyapunov method [32] based on single timescale optimization is no longer applicable. To address this issue, in the following sections, we develop a novel solution by constructing a two-timescale Lyapunov optimization framework. Specifically, using the deterministic upper bound on the Lyapunov drift-plus-penalty, we first decompose the long-term stochastic optimization problem into a series of deterministic instant problems, each of which is further decoupled into two subproblems in different timescales. After that, we propose an iterative algorithm integrating randomized rounding, Lagrangian method and convex optimization to solve these subproblems, respectively.

V. Two-Timescale Online Algorithm

To address this issue, in the following sections, we develop a two-timescale online optimization algorithm for joint microservice deployment, layer placement and loading, computing resource management and task scheduling (called OMLRC) to solve $\mathcal{P}_0$.

A. Problem Reformulation

It can be observed from $\mathcal{P}_0$ that the delay caused by the layer placement are on the large timescale, while those caused by task execution (including task computing and layer loading) are on the small timescale. To facilitate analysis, we evenly distribute the layer placement delay in each time frame t into all time slots within this frame. Then, the total delay of ES $m \in \mathcal{M}$ in serving tasks requesting microservice $s \in \mathcal{S}$ in time slot $\tau \in \mathcal{T}_t$ can be converted to

$$D_{m,s}^{\mathrm{tol}}(\tau) = D_m^{\mathrm{pev}}(\tau) + D_{m,s}^{\mathrm{exe}}(\tau) \tag{21}$$

where $D_m^{\mathrm{pev}}(\tau) = (D_m^{\mathrm{pla}}(t)/K)$ represents the layer placement delay in each time frame t evenly distributed into all $|\mathcal{T}_t| = K$ time slots, respectively. Substituting (21) to $\mathcal{P}_0$, we have

$$\begin{aligned} 2\mathcal{P}_1: \min_{\mathcal{J}_A(t),\mathcal{J}_B(\tau)} \ & \lim_{T\to\infty} \mathcal{F} \\ \text{s.t.} \ & (1),\ (2),\ (5),\ (7)-(12) \\ & \lim_{T\to\infty} \frac{1}{TK} \sum_{\tau=0}^{TK-1} D_{m,s}^{\mathrm{tol}}(\tau) \le \frac{D_s^{\mathrm{th}}}{K} \end{aligned} \tag{22}$$

where $\mathcal{F} = (1/TK) \sum_{\tau=0}^{KT-1} \sum_{i=1}^{I} F_{m,s}^{\mathrm{fail}}(D_{m,s}^{\mathrm{exe}}(\tau))$. Note that the reformulated problem $\mathcal{P}_1$ is equivalent to the original problem $\mathcal{P}_0$ with exactly the same optimization variables remaining in two different timescales, while all long-term metrics have been unified into a single timescale but will not affect the optimization performance.

Obviously, $\mathcal{P}_1$ is still a long-term stochastic optimization problem. The major challenges in solving problem $\mathcal{P}_1$ are 1) how the long-term average total delay constraint can be handled and and 2) how the two-timescale decision variables can be optimized simultaneously. To this end, in the next section, we employ the idea of Lyapunov optimization method [32] and modify it to accommodate the features of problem $\mathcal{P}_1$.

B. Problem Decomposition and Decoupling

First, we define a delay overflow queue to describe the deviation between the total delay for tasks requesting microservice s at ES m in time slot τ and the long-term delay budget. The dynamic evolution of such an overflow queue is as follows:

$$Q_m^s(\tau+1) = \left[D_{m,s}^{\mathrm{tol}}(\tau) - \frac{D_s^{\mathrm{th}}}{K} \right]^+ + Q_m^s(\tau) \tag{23}$$

with initial state $Q_m^s(0) = 0$.

Then, we introduce the quadratic Lyapunov function [32]

$$L(\mathbf{\Theta}(\tau)) \triangleq \frac{1}{2}\left[\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} Q_m^s(\tau)^2\right] \tag{24}$$

where $\mathbf{\Theta}(\tau) = [Q_1^1(\tau), \ldots, Q_M^S(\tau)]$. The Lyapunov function serves as a quantitative indicator of the congestion in all queues, and it is crucial to consistently strive for its minimization to ensure the stability of the queues. In accordance with [33], the conditional Lyapunov drift can be written as

$$\Delta(\mathbf{\Theta}(\tau)) = \mathbb{E}[L(\mathbf{\Theta}(\tau+K)) - L(\mathbf{\Theta}(\tau))|\mathbf{\Theta}(\tau)] \tag{25}$$

which measures the difference of the Lyapunov function between K consecutive time slots. Intuitively, by minimizing the Lyapunov drift in (25), we can prevent the queue backlogs from unbounded growth, and thus preserve $Q_m^s(\tau)$ to not violating the desirable constraints.

Accordingly, the Lyapunov drift-plus-penalty function can be expressed as

$$\Delta_V(\mathbf{\Theta}(\tau)) = \Delta(\mathbf{\Theta}(\tau)) + V\cdot\mathbb{E}[\mathcal{F}|\mathbf{\Theta}(\tau)] \tag{26}$$

where $V \in (0, +\infty)$ is a control parameter. The following theorem gives an analytical upper bound of such drift-plus-penalty in each time slot τ.

Theorem 1: Let $V \in (0, +\infty)$. For an arbitrary $\mathcal{J}_A(t), \mathcal{J}_B(\tau)$ $\forall s \in \mathcal{S}$ $\forall m \in \mathcal{M}$, the drift-plus-penalty is bounded under any possible decisions in any time slot τ

$$\begin{aligned}\Delta_V \le B + \sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} \mathbb{E}\Big\{Q_m^s(\tau)\big[D_{m,s}^{\text{tol}}(\tau)(\mathbf{\Theta}(\tau))\\ - D_s^{\text{th}}/K\big]|\mathbf{\Theta}(\tau)\Big\} + V\cdot\mathbb{E}[\mathcal{F}]\end{aligned} \tag{27}$$

where $B = (1/2)[D_{m,s}^{\text{tol}}(\max) - (D_s\text{th})/K]^2$ is a positive constant that adjusts the reliability and the satisfaction degree of the long-term total delay constraints.

Proof: By squaring both sides of the delay overflow queue in (23), we have

$$\begin{aligned}&Q_m^s(\tau+1)^2\\ &= \left[[D_{m,s}^{\text{tol}}(\tau) - D_s^{\text{th}}/K]^+ + Q_m^s(\tau)\right]^2\\ &\le \left[D_{m,s}^{\text{tol}}(\tau) - \frac{D_s^{\text{th}}}{K}\right]^2 + Q_m^s(\tau)^2 + 2Q_m^2(\tau)\left[D_{m,s}^{\text{tol}}(\tau) - \frac{D_s^{\text{th}}}{K}\right].\end{aligned} \tag{28}$$

By subtracting $Q_m^s(\tau)^2$ from both sides, and summing up all inequalities for $m \in \mathcal{M}, s \in \mathcal{S}$, we have

$$\begin{aligned}&\frac{1}{2}\sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\left[Q_m^s(\tau+1)^2 - Q_m^s(\tau)^2\right]\\ &\le \frac{1}{2}\sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\left\{\left[D_{m,s}^{\text{tol}}(\tau) - \frac{D_s^{\text{th}}}{K}\right]^2 + Q_i(\tau)\left[D_{m,s}^{\text{tol}}(\tau) - \frac{D_s^{\text{th}}}{K}\right]\right\}.\end{aligned} \tag{29}$$

Since $\rho_m^s(\tau)$ cannot exceed their upper bounds, combining (29) yields

$$\begin{aligned}&L(\Theta(\tau+1)) - L(\Theta(\tau))\\ &= \frac{1}{2}\sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\left[Q_m^i(\tau+1)^2 - Q_m^s(\tau)^2\right]\\ &\le \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\left\{\frac{1}{2}\left[D_{m,s}^{tol^{\max}} - \frac{D_s^{\text{th}}}{K}\right]^2 + Q_m^s(\tau)\left[D_{m,s}^{\text{tol}}(\tau) - \frac{D_s^{\text{th}}}{K}\right]\right\}.\end{aligned} \tag{30}$$

Finally, by adding $V\cdot[\mathcal{F}]$ to both sides of (30) and taking the expectation of both sides of $\mathbf{\Theta}(\tau)$, (27) can be derived. ■

Theorem 1 shows that the drift-plus-penalty is deterministically upper bounded in each time slot τ (i.e., the small timescale). Then, with slight mathematical manipulations, the upper bound of the drift-plus-penalty in each time frame t (i.e., the large timescale) can also be derived as

$$\begin{aligned}\Delta_V(\mathbf{\Theta}(t)) \le BK + \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\sum_{\tau\in\mathcal{T}_t}\mathbb{E}\{Q_m^s(\tau)[D_{m,s}^{\text{tol}}(\tau)\\ - D_s^{\text{th}}/K]|\mathbf{\Theta}(\tau)\} + V\cdot\sum_{\tau\in\mathcal{T}_t}\mathbb{E}[\mathcal{F}].\end{aligned} \tag{31}$$

Following the convention of the Lyapunov optimization method [32], [34], problem $\mathcal{P}_1$ can be transformed to opportunistically minimize the right-hand side of (31) subject to (1), (2), (5), and (7)–(12). In other words, the long-term stochastic optimization problem $\mathcal{P}_1$ can be decomposed into a series of deterministic instant problem $\mathcal{P}_2$, which is given by

$$\begin{aligned}\mathcal{P}_2: \min_{\mathcal{J}_A(t),\mathcal{J}_B(\tau)}\ & \mathcal{G}_{m,s}(t) = \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\sum_{\tau\in\mathcal{T}_t}\Big\{Q_m^s(t)[D_{m,s}^{\text{tol}}(\tau)\\ &- D_s^{\text{th}}/K]\Big\} + V\cdot\sum_{\tau\in\mathcal{T}_t}[\mathcal{F}]\\ \text{s.t.}\ & (1),\ (2),\ (5),\ (7)-(12).\end{aligned}$$

Note that decisions $\mathcal{J}_A(t) = \{\lambda_m^s(t), d_m^l(t)\}$ and $\mathcal{J}_B(\tau) = \{\vartheta_{m,m'}^l(\tau), \rho_m^s(\tau), x_{m,m'}^s(\tau)\}$ remain unchanged, and thus problem $\mathcal{P}_3$ is still a two-timescale MINLP. Different from the traditional MINLP problem with single timescale, decision variables of $\mathcal{P}_2$ are required to be iterated at two timescales. In Section VI, simulation results are provided to show that such a two-timescale iteration process can indeed converge, so that a near-optimal solution can be produced.

C. Algorithm Design for Large-Timescale Decisions

In order to obtain a near-optimal solution to $\mathcal{P}_2$, at the beginning of each time frame t, we propose an online algorithm that iteratively optimizes subproblems $\mathcal{P}_t$ and $\mathcal{P}_{\tau=tK}$.

1) Joint Microservice Manu Decision and Layer Placement Optimization in Time Frame t: Given the current backlogs of delay overflow queues for all type of tasks requesting different microservices at ES m, as well as the instantaneous reliability performance, the problem of making joint microservice manu decision and layer placement in each time frame t becomes

Algorithm 1: Procedure of JMLO

Initialize: At the beginning of time frame t, collect the state information of all microservice $s \in \mathcal{S}$ and ES $m \in \mathcal{M}$;
Linear relaxation: $\lambda_m^s(t) \in \{0,1\} \rightarrow \lambda_m^s(t) \in [0,1]$, $d_m^l(t) \in \{0,1\} \rightarrow d_m^l(t) \in [0,1]$;
Obtain $\{\tilde{\lambda}_m^s(t)\}$ and $\{\tilde{d}_m^l(t)\}$ through linear programming while satisfying the constraints (7) and (12);
for $s \in \mathcal{S}$ **do**
  Set $\widehat{\lambda}_m^s(t) = 1$ with the probability $\tilde{\lambda}_m^s(t)$;
for $m \in \mathcal{M}$ **do**
  Define $\tilde{\mathcal{L}}$ as the set of potential layers to be placed;
  if $\tilde{\mathcal{L}} = \emptyset$ **then**
    Set $\widehat{d}_m^l(t) = 1$ with the probability $\delta_i(t)$;
  else
    Set $\widehat{d}_m^l(t) = 1$ with the probability $\delta_i'(t)$;

Output: Solution of $\mathcal{P}_t$: $\widehat{\lambda}_m^s(t)$ and $\widehat{d}_m^l(t)$.

$$\mathcal{P}_t: \min_{\lambda_m^s(t), d_m^l(t)} \sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\left[D_m^{\text{pla}}(t) + D_m^{\text{loa}}(\tau)\right] + V \cdot \mathcal{F}$$
$$\text{s.t.} \quad (1),\ (7),\ (12).$$

In order to solve this integer programming problem $\mathcal{P}_4$, we adopt randomized rounding technique [35] and design a corresponding solution algorithm for joint microservice manu decision and layer placement optimization, called JMLO, which is summarized in Algorithm 1.

First, we relax the constraints of decision variables $\lambda_m^s(t)$ and $d_m^l(t)$ as

$$\lambda_m^s(t) \in \{0,1\} \rightarrow \lambda_m^s(t) \in [0,1] \tag{32}$$
$$d_m^l(t) \in \{0,1\} \rightarrow d_m^l(t) \in [0,1]. \tag{33}$$

Then, $\mathcal{P}_t$ can be solved in a polynomial time by using the linear programming solver [36], and we denote $\{\tilde{\lambda}_m^s(t)\}$ and $\{\tilde{d}_m^l(t)\}$ as the corresponding optimal solutions.

Our remaining issue is to round $\{\tilde{\lambda}_m^s(t)\}$ and $\{\tilde{d}_m^l(t)\}$ to obtain integer solutions, denoted by $\{\widehat{\lambda}_m^s(t)\}$ and $\{\widehat{d}_m^l(t)\}$. First, we round $\{\widehat{\lambda}_m^s(t)\}$ to 1 with probability $\{\tilde{\lambda}_m^s(t)\}$. Then, each ES can obtain the information about which microservices should be added in its microservice menu in time frame t. Based on $\{\widehat{\lambda}_m^s(t)\}$, $\{\widehat{d}_m^l(t)\}$ can be determined as follows. For each ES $m \in \mathcal{M}$, denote $\tilde{\mathcal{L}}$ by the set of potential layers need to be placed. If $\tilde{\mathcal{L}} = \emptyset$, ES m will load the required layers with a probability $\delta_{m,s}^l(t)'$ given by

$$\delta_{m,s}^l(t) = \begin{cases} 1, & \text{if } \tilde{\lambda}_m^s(t) \geq \prod_{l \in \tilde{\mathcal{L}}}\left(1 - \tilde{d}_m^l(t)\right) \\ \dfrac{\tilde{\lambda}_m^s(t)}{\prod_{l \in \tilde{\mathcal{L}}}\left(1 - \tilde{d}_m^l(t)\right)}, & \text{else.} \end{cases}$$

Otherwise, ES m will place the layer $l \in \tilde{\mathcal{L}}$ downloaded from the cloud repository with the probability $\delta_{m,s}^l(t)' = ([\tilde{\lambda}_m^s(t)]/[\tilde{d}_m^l(t)])$.

Lemma 1: The gap between the solution returned by JMLO, denoted by $\widehat{\mathcal{P}}_t$, and the optimal solution, denoted by $\mathcal{P}_t^*$, is bounded by

$$\widehat{\mathcal{P}}_t - \mathcal{P}_t^* \leq \Lambda \tag{34}$$

where $\Lambda = \sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\{\sum_{l \in \mathcal{L}} \sum_{m \in \mathcal{M}} (v_l / [r_{m,c}^{\text{pla}}(t)]) + D_m^{\text{loa}}(\tau)\} + V$.

Proof: Taking subtraction between $\widehat{\mathcal{P}}_t$ and $\mathcal{P}_t^*$, and by some simple mathematical manipulation, we have

$$\begin{aligned} &\widehat{\mathcal{P}}_t - \mathcal{P}_t^* \\ &= \sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\left[\widehat{D}_m^{\text{pla}}(t) + D_m^{\text{loa}}(\tau)\right] + V\widehat{\mathcal{F}} \\ &\quad - \left\{\sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\left[D_m^{\text{pla}}(t)^* + D_m^{\text{loa}}(\tau)\right] + V \cdot \mathcal{F}^*\right\} \\ &= \sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\left\{\sum_{l \in \mathcal{L}} \frac{v_l}{r_{m,c}(t)}\left[\widehat{d}_m^l(t) - d_m^l(t)^*\right]\right. \\ &\quad \left. + D_m^{\text{loa}}(\tau) + D_i^{\text{tra}}(\tau)\right\} + V\left(\widehat{\mathcal{F}} - \mathcal{F}^*\right). \end{aligned} \tag{35}$$

Since $\lambda_m^s(t)$, $d_m^l(t)$ and $\mathcal{F}$ all belong to $[0,1]$, we have

$$\begin{aligned} &\widehat{\mathcal{P}}_t - \mathcal{P}_t^* \\ &\leq \sum_{\tau \in \mathcal{T}_t} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)\left\{\sum_{l \in \mathcal{L}} \frac{v_l}{r_{m,c}(t)} + \left[D_m^{\text{loa}}(\tau)\right.\right. \\ &\quad \left.\left. + D_i^{\text{tra}}(\tau)\right]\right\} + V = \Lambda \end{aligned} \tag{36}$$

and Λ is known at the beginning of time frame t. ■

Lemma 1 specifies that JMLO can reach asymptotically optimal solutions for microservice manu and layer placement decision for all microservices and ESs in each time frame t. These solutions will be used in the following Section to further determine the optimal layer loading, task scheduling and resource allocation in time slot $\tau = tK$.

2) Layer Loading Optimization in Time Slot $\tau = tK$: In this section, we fixed microservice manu decision $\widehat{\lambda}_m^s(t)$ and layer placement decision $\widehat{d}_m^l(t)$, and then the subproblem to decide layer loading $\vartheta_{m,m'}^l(tK)$ at the first time slot in frame t, i.e., $\tau = tK$, as shown below

$$\mathcal{P}_{tK}': \min_{\vartheta_{m,m'}^l(tK)} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(tK) D_m^{\text{loa}}(tK) + V\mathcal{F}$$
$$\text{s.t.} \quad (9)-(12). \tag{37}$$

By relaxing the integer decision variable $\vartheta_{m,m'}^l(tK) \in [0,1]$, problem $\mathcal{P}_{tK}'$ becomes a linear program (LP) problem. First, we define its partial Lagrangian function as

$$\begin{aligned} &\mathcal{L}\left(\vartheta_{m,m'}^l(tK), \mu_1(tK), \mu_2(tK), \mu_3(tK)\right) \\ &= \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(tK) D_m^{\text{loa}}(tK) + V\mathcal{F} + \mu_1(tK)[d_{m'}^l(t) \\ &\quad - \vartheta_{m,m'}^l(tK)] + \mu_2(tK)\left[1 - \sum_{m' \in \mathcal{M}} \vartheta_{m,m'}^l(tK)\right] + \mu_3(tK) \\ &\quad \left[d_m^l(t) + \sum_{m' \in \mathcal{M}} \vartheta_{m,m'}^l(tK) - \lambda_m^s(t)\varpi_s^l\right] \end{aligned} \tag{38}$$

where $\mu_1(tK), \mu_2(tK), \mu_3(tK) > 0$ is the Lagrangian multipliers. Set $G\mu_1(tK), \mu_2(tK), \mu_3(tK)) =$

Algorithm 2: Iterative Algorithm OMLRC

Initialize: Observe the instantaneous queue set $\mathbf{\Theta}(t)$, task generation $\alpha_m^s(\tau)$ and ESs' lifetime $D_m^{\text{life}}(\tau)$;
for *each time frame* t **do**
  Set the initial iteration index $j = 1$;
  while $|\mathcal{G}_{m,s}^{j'}(t) - \mathcal{G}_{m,s}^{j'-1}(t)| > \varphi'$ **do**
    for $s \in \mathcal{S}$, $m \in \mathcal{M}$ **do**
      Obtain $\widehat{\lambda}_m^s(t)$ and $\widehat{d}_m^l(t)$ by **Algorithm 1**;
      for *time slot* $\tau = tK$ **do**
        With fixed $\widehat{\lambda}_m^s(t)$ and $\widehat{d}_m^l(t)$, obtain $\widehat{\vartheta}_{m,m'}^l(tK)$, $\widehat{x}_{m,m'}^s(tK)$ and $\widehat{\rho}_m^s(tK)$ by solving problem $\mathcal{P}'_{tK}$ and $\mathcal{P}''_{tK}$, respectively;
      Update the value of delay overflow queue $Q_m^s(t)$;
    Update the iteration index $j = j + 1$;

Output: $\lambda_m^s(t)^\dagger$, $d_m^l(t)^\dagger$, $\vartheta_{m,m'}^l(tK)^\dagger$, $x_{m,m'}^s(tK)^\dagger$, $\rho_m^s(tK)^\dagger$.

$([\partial \mathcal{L}(\vartheta_{m,m'}^l(tK), \mu_1(tK), \mu_2(tK), \mu_3(tK))]/[\partial \vartheta_{m,m'}^l(tK)])$, which can be derived as

$$\begin{aligned} &G(\mu_1(tK), \mu_2(tK), \mu_3(tK)) \\ &= Q_i(tK) x_i^m(t) \frac{\partial D_m^{\text{loa}}(tK)}{\partial \vartheta_{m,m'}^l(tK)} + V \frac{\partial \mathcal{F}}{\partial \vartheta_{m,m'}^l(tK)} \\ &\quad - \mu_1(tK) - \mu_2(tK) + \mu_3(tK). \end{aligned} \tag{39}$$

Note that, $([\partial G(\mu_1(tK), \mu_2(tK), \mu_3(tK))]/[\partial \vartheta_{m,m'}^l(tK)]) \geq 0$. Hence, problem $\mathcal{P}'_{tK}$ is a convex linear optimization problem, which can be solved by Gradient Descent method [37]. Finally, set the obtained solution $\widehat{\vartheta}_{m,m'}^l(tK)$ to $\{0, 1\}$ by a rounding manner.

Lemma 2: Similar to the Lemma 1, the gap between the solution returned by solving $\mathcal{P}'_{tK}$, denoted by $\widehat{\mathcal{P}}'_{tK}$, and the optimal solution, denoted by $\mathcal{P}'^*_{tK}$, is bounded by

$$\widehat{\mathcal{P}}'_{tK} - \mathcal{P}'^*_{tK} \leq \Lambda' \tag{40}$$

where $\Lambda' = \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t)(v_l/[r_{m,m'}(t)]) + V$.

3) Task Scheduling and Resource Allocation Optimization in Time Slot $\tau = tK$: Similar to problem $\mathcal{P}'_{tK}$, the optimal computing resource allocation for each MD i at time slot $\tau = tK$ can be obtained by solving the following problem:

$$\begin{aligned} \mathcal{P}''_{tK}: \min_{x_{m,m'}^s(\tau), \rho_m^s(\tau)} \; & \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(tK)[D_{m,s}^{\text{com}}(tK) \\ & + D_{m,s}^{\text{sch}}(tK)] + V\mathcal{F} \\ \text{s.t.} \quad & (2),\ (1),\ (5). \end{aligned} \tag{41}$$

It is not difficult to verify that $\mathcal{P}''_{tK}$ is a convex problem, and thus standard convex algorithms can be used.

4) Overall Iterative Optimization-Based Algorithm: The overall iterative optimization-based algorithm OMLRC is described in Algorithm 2. The key idea is to iteratively optimize microservice manu decision $\lambda_m^s(t)^\dagger$, layer placement $d_m^l(t)^\dagger$, layer loading $\vartheta_{m,m'}^l(tK)^\dagger$, task scheduling $x_{m,m'}^s(tK)^\dagger$ and resource allocation $\rho_m^s(tK)^\dagger$, respectively.

D. Algorithm Design for Small-Timescale Decisions

In this section, the subproblem to decide layer loading $\vartheta_{m,m'}^l(\tau)$, task scheduling $x_{m,m'}^s(\tau)$ and computing resource allocation $\rho_m^s(\tau)$, $\tau \in \mathcal{T}_t$, $\tau \neq tK$ as shown below

$$\begin{aligned} \mathcal{P}_{\tau \neq tK}: \min_{\mathcal{J}_B(\tau \neq tK)} \; & \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(tK) D_i^{\text{exe}}(\tau) + V \cdot \mathcal{F} \\ \text{s.t.} \quad & (1),\ (2),\ (5),\ (9)-(12). \end{aligned} \tag{42}$$

For time slot $\tau \in \mathcal{T}_t$, $\tau \neq tK$, the microservice manu decision $\lambda_m^s(t)$ and layer placement $d_m^l(t)$ are deterministic, and the algorithm for solving $\mathcal{P}_{\tau \neq tK}$ is similar to $\mathcal{P}'_{tK}$ and $\mathcal{P}''_{tK}$.

E. Summary of OMLRC

In summary, the proposed OMLRC consists of the problem reformulation, problem decomposition, joint microservice manu and layer placement optimization and joint layer loading, task scheduling and resource allocation optimization, as described in Sections V-A–V-D. The implementation of OMLRC is detailedly illustrated in Algorithm 2. Unlike the performance analysis of single timescale algorithms, we consider a more comprehensive performance analysis of OMLRC on two different timescales.

Theorem 2 (Computational Complexity): The computational complexity of the proposed OMLRC is $O[T \cdot [(LS)^{2.055} + 3jK]]$, where T is the number of time frames, K is the number of time slots within each frame, and j is the average number of iterations.

Proof: The complexity of OMLRC depends on the alternate iterations of JMLO and Lagrangian method in each time frame. As stated in [35], applying the linear relaxation and linear programming solver for JMLO has an asymptotic computation time complexity of $O((LS)^{2.055})$, where S and L represent the number of candidate choices of $\lambda_m^s(t)$ and $d_m^l(t)$, respectively. Furthermore, according to [37], the complexity of Lagrangian and convex method is in the linear order with the number of decision variables and iterations. Then, the computational complexity of the proposed OALLC in each frame can be expressed as $O[T \cdot [(LS)^{2.055} + 3jK]]$. ■

Theorem 3 (Optimality): Given Lyapunov control parameter V, the optimality gap between the solution obtained by the proposed OMLRC and the theoretically optimal solution to problem $\mathcal{P}_0$ can be expressed as

$$\frac{1}{KT} \sum_{\tau \in \mathcal{T}_t} \sum_{t=0}^{t-1} \mathbb{E}[\mathcal{F}] \leq \varepsilon + \frac{B}{V} + \frac{\Lambda + \Lambda'}{VT} \tag{43}$$

where ε represents the theoretically optimal solution, and B is defined in (27). Obviously, when the control parameter V is large enough, the solution obtained by OMLRC can approach the optimal solution ε infinitely.

Proof: Based on Theorem 1 and Lemmas 1 and 2, (26) can be scaled as

$$\begin{aligned} &\Delta_V(\mathbf{\Theta}(t)) \\ &= \Delta(\mathbf{\Theta}(t)) + V \cdot \mathbb{E}[\mathcal{F}] \\ &\leq BK + \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} \sum_{\tau \in \mathcal{T}_t} \mathbb{E}\left\{ Q_m^s(\tau) \left[D_{m,s}^{\text{tol}}(\tau)^* - \frac{D_s^{\text{th}}}{K} \right] \middle| \mathbf{\Theta}(\tau) \right\} \end{aligned}$$

$$+ V \sum_{\tau \in \mathcal{T}_t} \mathbb{E}[\mathcal{F}]$$
$$\leq BK + V \cdot K\varepsilon + \Lambda + \Lambda' \quad (44)$$

where ε is the theoretically optimal solution.

Then, summing up (44) over T time frames, we can get

$$\begin{aligned}
&\big(B + V \cdot \varepsilon + (\Lambda + \Lambda')/K\big) \cdot KT \\
&\geq \sum_{t=0}^{T-1} \mathbb{E}[\Delta_V(\boldsymbol{\Theta}(\tau))|\boldsymbol{\Theta}(\tau)] \\
&= \mathbb{E}[L(\boldsymbol{\Theta}(KT)) - L(\boldsymbol{\Theta}(0))] + V \sum_{\tau \in \mathcal{T}_t} \sum_{t=0}^{T-1} \mathbb{E}[\mathcal{F}] \\
&= \mathbb{E}[L(\boldsymbol{\Theta}(KT))] + V \sum_{\tau \in \mathcal{T}_t} \sum_{t=0}^{T-1} \mathbb{E}[\mathcal{F}] - \mathbb{E}[L(\boldsymbol{\Theta}(0))]. \quad (45)
\end{aligned}$$

Finally, by moving $\mathbb{E}[L(\boldsymbol{\Theta}(0))]$ to the left-hand side of (45), and dividing both sides by V and T, Theorem 3 can be obtained. ■

VI. Simulation Results

A. Evaluation Setup

Consider a MATLAB-based simulation environment for an edge computing system with $M = 10$ ESs and $S = 15$ types of microservices. Each frame can be further regarded as a combination of $K = 10$ fine-grained time slots, where the length of each slot is $\gamma = 10s$. Each ES has a maximum caching capacity of $\Omega_m^{ca} = 50Gb$, as well as a CPU computing frequency of $f_m^{es} = 20$ GHz. Table II lists the values of main simulation parameters. Similar settings have also been employed in [9], [15], [23], [29], [38], [39], and [40]. Furthermore, to show the superiority of the proposed OMLRC, the performances of the following schemes are also evaluated as benchmarks.

1) *JMDLS [7]:* The layer placement, layer loading and computing resource allocation are jointly optimized to improve the edge throughput. However, this scheme is executed synchronously in a single timescale, and the dynamic task generations and uncertain lifetime of ESs are ignored.
2) *O2TL [41]:* The layer placement is decided in the large timescale, and computing resource allocation are decided in the small timescale. However, this scheme ignores the layer sharing and task scheduling.
3) *EGO [14]:* The dynamic task generations and uncertain lifetime of ESs are modeled, and computing resource allocation is optimized to improve the reliability. However, this scheme is executed in a single timescale and does not enable layer sharing.

B. Performance Evaluations

In Fig. 3, we evaluate the convergence property of the proposed OMLRC in solving problem $\mathcal{P}_0$ while varying the value of parameter V from 10^6 to 8×10^6. With time elapses, OMLRC demonstrates quick convergence for each value of V, leading to a stable average service reliability. This verifies the stability of the proposed algorithm. The control parameter V is used to adjust the influence of the Lyapunov function on the system state, balancing the importance between the long-term constraint (i.e., total delay) and the optimization objective (i.e., the average reliability). An increase in V indicates higher reliability requirements for the whole system, resulting in a higher reliability value under the control of the proposed algorithm and faster convergence to the optimal solution. As shown in this figure, there is a gap in the convergent optimal solution of the proposed algorithm under the three different V. This gap occurs because the values of V differ significantly, guiding the network operator to choose the appropriate V according to their personalized requirements in the practical implementation.

TABLE III
Simulation Parameters

Parameter	Value	Parameter	Value
$\alpha_m^s(\tau)$	$[20, 50]$	$r_{m,m'}^{loa}(t)$	$[10, 20]$ Mb/s
v_l	$[1, 3]$ Gb	β_s	300 cycles/bit
$r_{m,c}^{pla}(t)$	5 Mb/s	$C_m(\tau)$	$[0.01, 0.5]$
v_s	$[0.5, 2]$ Mb	S	10

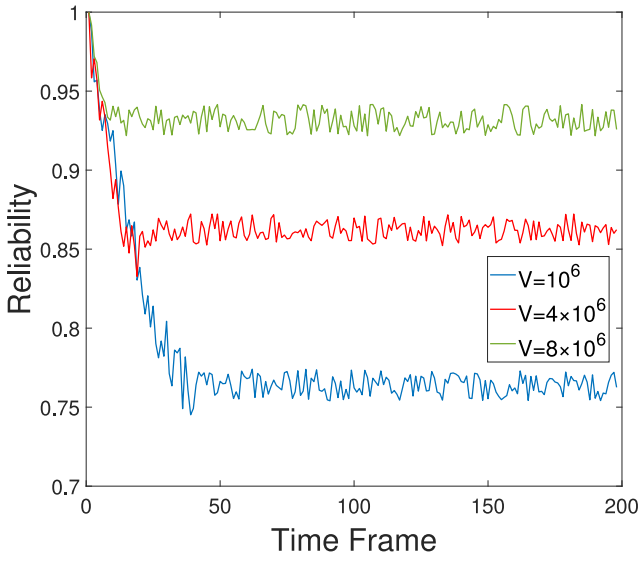

Fig. 3. Convergence of solving problem $\mathcal{P}_0$.

Figs. 4 and 5 show the performance comparison of execution delay with respect to the transmission rate of cloud-ES and ES-ES. In Fig. 4, as the transmission rate of ES-cloud increases, the execution delay under all algorithms decrease, indirectly improving reliability. This is because ESs can download the required layers from the cloud repository at a faster rate, which greatly decreases the layer placement delay $D_m^{\text{pla}}(t)$. In Fig. 5, as the transmission rate of ES-ES increases, the execution delay under JMDLS and OMLRC decrease and remains under EGO and O2TL. This is because each ES can loading the required layers from other ESs at a faster rate, which greatly increases the layer loading delay $D_m^{\text{loa}}(\tau)$. EGO and O2TL are independent of layer loading, and thus remain constant. Furthermore, these figures demonstrate that EGO and O2TL have limited control over

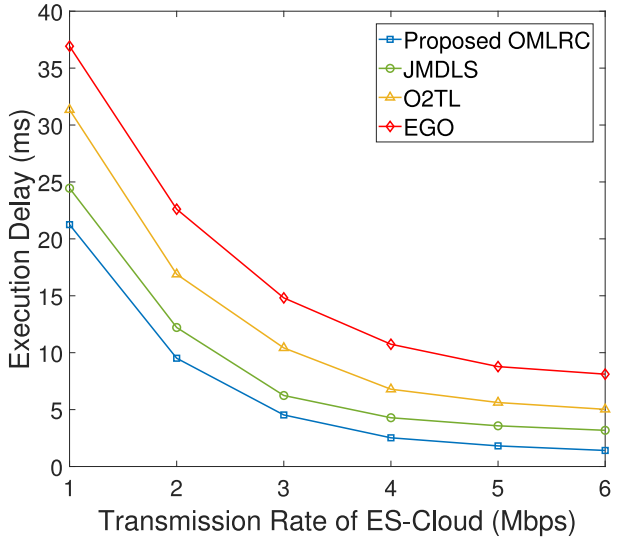

Fig. 4. Execution delay w.r.t. $r^{\text{pla}}_{m,c}(\tau)$.

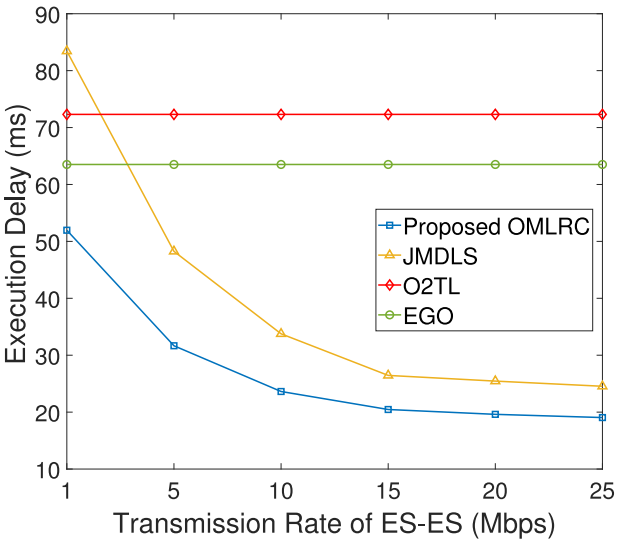

Fig. 5. Execution delay w.r.t. $r^{\text{loa}}_{m,m'}(\tau)$.

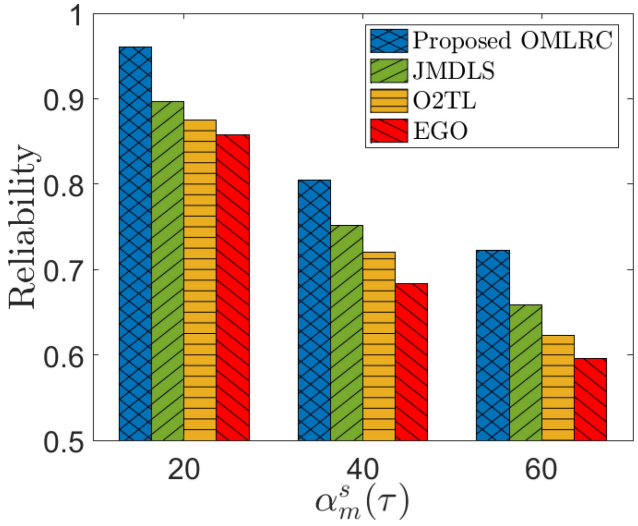

Fig. 6. Comparison on service reliability with amount of tasks $\alpha^s_m(\tau)$.

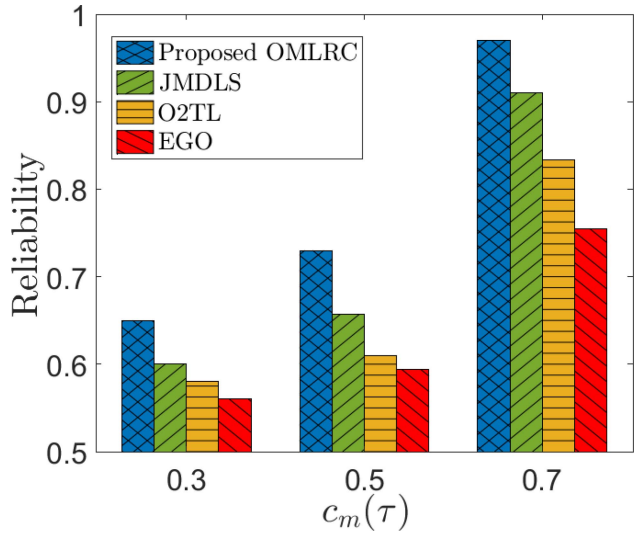

Fig. 7. Comparison on service reliability with $c_m(\tau)$ of ESs.

execution delay, primarily due to their neglect of container layer sharing, resulting in frequent preparation of common layers in a single ES and consequently increasing the delay in layer placement and loading. In contrast, the proposed OMLRC effectively reduces execution delay by employing a two-timescale control mechanism for layer placement and layer loading. This approach proves to be efficient in achieving a substantial decrease in execution delay by about 28.57%.

Fig. 6 shows the performance comparison of reliability with respect to the amount of tasks $\alpha^s_m(\tau)$. In this figure, as the amount of tasks $\alpha^s_m(\tau)$ increases, the reliability decreases correspondingly. This trend can be attributed to the fact that with a larger amount of tasks $\alpha^s_m(\tau)$, tasks requesting microservices may be more difficult executed within the ESs' lifetime range. In addition, we can see from these figures that EGO and O2TL have a poor control effect on reliability, which is due to its neglect of layer loading among ESs. Although JMDLS considers layer loading, but its single timescale strategy also fails to show advantages in the dynamic system. Compared with the three benchmarks, the proposed OMLRC increases the reliability by about 12.4% due to its two-timescale control over layer placement and layer loading for each ES along with the joint management of task scheduling and resource allocation, in an online manner.

Fig. 7 shows the performance comparison of service reliability with respect to $c_m(\tau)$. In this figure, as $c_m(\tau)$ increase, the service reliability increase correspondingly. This is because the increase of $c_m(\tau)$ means that the lifetime of ES m increases, so that the microservices deployed on ES m can handle more computation tasks, which improves the service reliability. In addition, this figure illustrates that the reliability of the proposed algorithm has the best control effect. This is because although JMDLS, O2TL and EGO consider intra-ES sharing by layer placement, they ignore the importance of inter-ESS sharing by layer loading. When the lifetime of ESs increases, the proposed scheme can deploy more diverse microservices to handle more diverse and larger number of

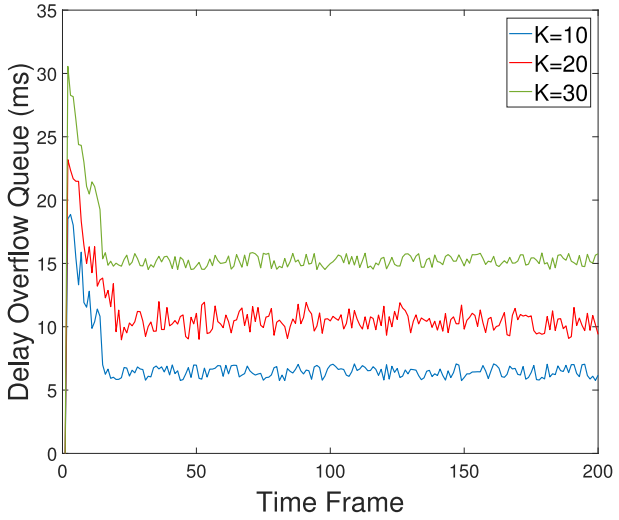

Fig. 8. Performance of queues backlog with different K.

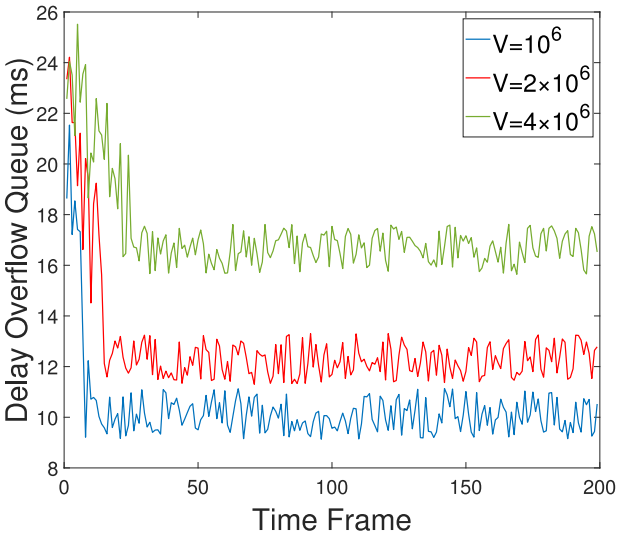

Fig. 9. Performance of queues backlog with different V.

in the case of limited cache capacity, thereby improving the system-wide service reliability.

In Fig. 8, we evaluate the impact of parameter K on the average delay overflow queue backlog over time. We set $V = 4\times10^6$ and $K = 10, 20, 30$. As shown in Fig. 8, increasing the value of K leads to larger backlogs in both the delay overflow queues and longer convergence times. This is because, as the time frame length K increases, the layer placement $d_m^l(t)$ and layer loading $\vartheta_{m,m'}^l(\tau)$ hence cannot be updated in time, and computing resource allocation is difficult to adapt the computing requirement of tasks (which change dynamically in each time slot $\tau \in \mathcal{T}_t$). Intuitively, the values of the average delay overflow queue backlog also gradually become stable over time and fluctuate slightly in a fixed value range. Note that the value of K can be set according to the maximum tolerant delay of the edge network. For example, we can see from Fig. 8 that, when we set $K = 20$, the backlog of delay overflow queue can be reduced by 60% compared with the case when $K = 10$.

TABLE IV
COMPUTATIONAL COMPLEXITY (IN MILLISECOND) WITH DIFFERENT NUMBER OF TASKS $\alpha_m^s(\tau)$

Benchmarks \ $\alpha_m^s(\tau)$	30	40	50	60
Proposed OMLRC	15.04	26.09	34.11	44.21
JMDLS	7.12	12.33	16.56	18.39
O2TL	45.33	50.16	80.18	108.64
EGO	11.65	24.84	33.76	48.09

In Fig. 9, we evaluate the impact of parameter V on the delay overflow queue backlog over time. The timeline is divided into $T = 200$ time frames and each frame has $K = 10$ time slots, and the value of V ranges from 10^6 to 8×10^6. We can see from Fig. 9 that the average delay overflow queue backlog values gradually stabilize over time as V increases. With a higher value of V, the edge computing system aims for increased reliability by placing more layers and utilizing fewer ESs to process microservice requests. However, this can result in an increase in the backlog of the delay overflow queue $Q_i(\tau)$. In the initial 30 time frames, both queues experience fluctuations because the system has just started cold, and as time goes on, the system accumulates sufficient queue information, gradually balancing the relationship between system-wide reliability $\mathcal{R}$ and backlogs of $Q_i(\tau)$.

Table IV examines the computational complexity of the proposed offloading scheme and the benchmark schemes in terms of the average running time (in millisecond) on a PC with 3.6-GHz Intel Core i7 CPU and 32-GB RAM with respect to different amount of tasks $\alpha_m^s(\tau)$. This table illustrates that the running time of all schemes increase with the number of tasks. JMDLS and EGO have shorter running times because they only consider synchronous execution of a single time scale. JMDLS further ignores the dynamic nature of task generation and ES lifetime, which reduces computational complexity but results in less control over microservices' reliability in asynchronous and dynamic environments, which has also been shown in Figs. 6 and 7. The proposed scheme's running time is better than O2TL, despite also considering dual time scale decisions. This is because O2TL only considers layer sharing within the ES, causing resource-constrained ES to incur high-layer placement and task execution overhead. It is worth noting that this computation analysis indicates the practicality of the proposed OMLRC, as it can not only achieve the best performance but also work a in comparable time complexity.

VII. CONCLUSION

This article delves into a novel reliability-enhanced microservice deployment problem tailored for edge computing with layer sharing. By establishing a quantifiable relationship between system-wide edge computing service reliability and the lifetime of ESs, we present a comprehensive investigation of a two-timescale joint optimization encompassing crucial aspects, such as layer placement and loading, task scheduling, and computing resource allocations across multiple ESs. To address this intricate problem, we propose an online algorithm, which leverages an improved Lyapunov technique

integrating randomized rounding, Lagrangian method and convex optimization. Theoretical analyses reveal that the proposed solution can converge to asymptotic optimum with a low complexity. Extensive simulations demonstrate that our proposed solution exhibits remarkable enhancements, with an increase of 12.4% in reliability and a reduction of 28.57% in latency, compared to state-of-the-art benchmarks.

REFERENCES

[1] L. Su, N. Wang, R. Zhou, and Z. Li, "Dynamic service placement and request scheduling for edge networks," *Comput. Netw.*, vol. 213, Aug. 2022, Art. no. 108997.

[2] J. Chen, C. Yi, S. D. Okegbile, J. Cai, and X. Shen, "Networking architecture and key supporting technologies for human digital twin in personalized healthcare: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 1, pp. 706–746, 1st Quart., 2024.

[3] 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.

[4] R. Chen, W.-X. Long, G. Mao, and C. Li, "Development trends of mobile communication systems for railways," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3131–3141, 4th Quart., 2018.

[5] J. Lou, H. Luo, Z. Tang, W. Jia, and W. Zhao, "Efficient container assignment and layer sequencing in edge computing," *IEEE Trans. Services Comput.*, vol. 16, no. 2, pp. 1118–1131, Mar./Apr. 2023.

[6] Y. Yuan, C. Yi, B. Chen, Y. Shi, and J. Cai, "A computation offloading game for jointly managing local pre-processing time-length and priority selection in edge computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 9, pp. 9868–9883, Sep. 2022.

[7] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, "Layer-aware collaborative microservice deployment toward maximal edge throughput," in *Proc. IEEE Conf. Comput. Commun.(INFOCOM)*, 2022, pp. 71–79.

[8] J. Li, C. Yi, J. Chen, K. Zhu, and J. Cai, "Joint trajectory planning, application placement, and energy renewal for UAV-assisted MEC: A triple-learner-based approach," *IEEE Internet Things J.*, vol. 10, no. 15, pp. 13622–13636, Aug. 2023.

[9] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, "Layer aware microservice placement and request scheduling at the edge," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–9.

[10] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy docker containers," in *Proc. 14th USENIX FAST*, 2016, pp. 181–195.

[11] C. Zheng et al., "Wharf: Sharing docker images in a distributed file system," in *Proc. ACM SOCC*, 2018, pp. 174–185.

[12] J. Liu et al., "Reliability-enhanced task offloading in mobile edge computing environments," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10382–10396, Jul. 2022.

[13] L. Zhao et al., "Joint coverage-reliability for budgeted edge application deployment in mobile edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3760–3771, Dec. 2022.

[14] K. Cao, Y. Cui, Z. Liu, W. Tan, and J. Weng, "Edge intelligent joint optimization for lifetime and latency in large-scale cyber–physical systems," *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22267–22279, Nov. 2022.

[15] R. Zhou, X. Wu, H. Tan, and R. Zhang, "Two time-scale joint service caching and task offloading for UAV-assisted mobile edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2022, pp. 1189–1198.

[16] H. Qiu, H. Noura, M. Qiu, Z. Ming, and G. Memmi, "A user-centric data protection method for cloud storage based on invertible DWT," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1293–1304, Oct.–Dec. 2021.

[17] X. Zhou, S. Ge, T. Qiu, K. Li, and M. Atiquzzaman, "Energy-efficient service migration for multi-user heterogeneous dense cellular networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 890–905, Feb. 2023.

[18] Y. Guo and W. Yao, "A container scheduling strategy based on neighborhood division in micro service," in *Proc. IEEE/IFIP NOMS*, 2018, pp. 1–6.

[19] W. Lv et al., "Graph-reinforcement-learning-based dependency-aware microservice deployment in edge computing," *IEEE Internet Things J.*, vol. 11, no. 1, pp. 1604–1615, Jan. 2024.

[20] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Auto-scaling containerized applications in geo-distributed clouds," *IEEE Trans. Services Comput.*, vol. 16, no. 6, pp. 4261–4274, Nov./Dec. 2023.

[21] M. Adeppady, P. Giaccone, H. Karl, and C. F. Chiasserini, "Reducing microservices interference and deployment time in resource-constrained cloud systems," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 3, pp. 3135–3147, Sep. 2023.

[22] 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.*, early access, Nov. 10, 2023, doi: 10.1109/TNET.2023.3330255.

[23] 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. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–9.

[24] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for DNN inference tasks in mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11453–11464, Jul. 2023.

[25] C.-W. Lee, M.-C. Chuang, M. C. Chen, and Y. S. Sun, "Seamless handover for high-speed trains using femtocell-based multiple egress network interfaces," *IEEE Trans. Wireless Commun.*, vol. 13, no. 12, pp. 6619–6628, Dec. 2014.

[26] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with KubeEdge," in *Proc. IEEE/ACM SEC*, 2018, pp. 373–377.

[27] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–35, 2022.

[28] A. A. Abud, F. Le Goff, and G. Avolio, "Performance evaluation of distributed file systems for the phase-II upgrade of the ATLAS experiment at CERN," *J. Phys. Conf. Ser.*, vol. 1525, no. 1, 2020, Art. no. 012028.

[29] C. Ying, Z. Zhao, C. Yi, Y. Shi, and J. Cai, "An AoTI-driven joint sampling frequency and access selection optimization for industrial wireless sensor networks," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 12311–12325, Sep. 2023.

[30] A. J. Fernandez, "Tolerance limits for k-out-of-n systems with exponentially distributed component lifetimes," *IEEE Trans. Reliab.*, vol. 59, no. 2, pp. 331–337, Jun. 2010.

[31] R. Chen, C. Yi, K. Zhu, B. Chen, J. Cai, and M. Guizani, "A three-party hierarchical game for physical layer security aware wireless communications with dynamic trilateral coalitions," *IEEE Trans. Wireless Commun.*, early access, Oct. 16, 2023, doi: 10.1109/TWC.2023.3322776.

[32] M. Neely, *Stochastic Network Optimization With Application to Communication and Queueing Systems*. San Rafael, CA, USA: Morgan Claypool Publ., 2010.

[33] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, no. 1, pp. 1–144, 2006.

[34] Y. Shi, C. Yi, R. Wang, Q. Wu, B. Chen, and J. Cai, "Service migration or task rerouting: A two-timescale online resource optimization for MEC," *IEEE Trans. Wireless Commun.*, vol. 23, no. 2, pp. 1503–1519, Feb. 2024.

[35] A. Srinivasan, "Approximation algorithms via randomized rounding: A survey," in *Proc. Adv. Topics Math. (PWN)*, 1999, pp. 9–71.

[36] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *Proc. 56th IFOCS*, pp. 230–249, 2015.

[37] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[38] L. Yu, Z. Li, J. Liu, and R. Zhou, "Online and energy-efficient task-processing for distributed edge networks," *Comput. Netw.*, vol. 193, Jul. 2021, Art. no. 107875.

[39] Y. Shi, C. Yi, B. Chen, C. Yang, K. Zhu, and J. Cai, "Joint online optimization of data sampling rate and preprocessing mode for edge-cloud collaboration enabled industrial IoT," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16402–16417, Sep. 2022.

[40] Y. He et al., "Two-timescale resource allocation for automated networks in IIoT," *IEEE Trans. Wireless Commun.*, vol. 21, no. 10, pp. 7881–7896, Oct. 2022.

[41] X. Li, X. Zhang, and T. Huang, "Asynchronous online service placement and task offloading for mobile edge computing," in *Proc. 18th IEEE SECON*, 2021, pp. 1–9.

You Shi received the M.S. degree from the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China, in 2020. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

His main research interests include mobile edge computing, online optimization, service deployment, resource management, Internet of Things, and 5G and beyond.

Yuye Yang received the B.S. degree from the School of Nanjing University of Aeronautics and Astronautics, Nanjing, China, where he is currently pursuing the M.S. degree with the College of Computer Science and Technology.

His main research interests include mobile edge computing, digital twin, stochastic optimization, resource management, and online learning.

Changyan Yi (Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, in 2018.

He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. From September 2018 to August 2019, he was a Research Associate with the University of Manitoba. His research interests include stochastic optimization, mechanism design, game theory, queueing scheduling, and machine learning with applications in resource management and decision making for various networking systems and services.

Bing Chen received the B.S. and M.S. degrees in computer engineering from Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 1992 and 1995, respectively, and the Ph.D. degree from the College of Information Science and Technology, NUAA in 2008.

He has been with NUAA since 1998, where he is currently a Professor with the Computer Science and Technology Department. His main research interests include cloud computing, wireless communications, and cognitive radio networks.

Jun Cai (Senior Member, IEEE) received the Ph.D. degree from the University of Waterloo, Waterloo, ON, Canada, in 2004.

From June 2004 to April 2006, he was a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellow with McMaster University, Hamilton, ON, Canada. From July 2006 to December 2018, he has been with the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, where he was a Full Professor and the NSERC Industrial Research Chair. Since January 2019, he was a Full Professor with the Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada, and the PERFORM Centre Research Chair. His current research interests include edge/fog computing, ehealth, radio resource management in wireless communication networks, and performance analysis.

Dr. Cai received the Best Paper Award from Chinacom in 2013, the Rh Award for Outstanding Contributions to Research in Applied Sciences in 2012 from the University of Manitoba, and the Outstanding Service Award from IEEE Globecom 2010. He served as the Technical Program Committee (TPC) Co-Chair for IEEE GreenCom 2018; the Track/Symposium TPC Co-Chair for the IEEE VTC-Fall 2019, IEEE CCECE 2017, IEEE VTC-Fall 2012, IEEE Globecom 2010, and IWCMC 2008; the Publicity Co-Chair for IWCMC 2010, 2011, 2013, 2014, 2015, 2017, and 2020; and the Registration Chair for QShine 2005. He also served on the editorial board for IEEE INTERNET OF THINGS JOURNAL, *IET Communications*, and *Wireless Communications and Mobile Computing*.