Layer-Aware Microservice Deployment for Edge Computing with Service Reliability Provisioning

You Shi[1], Yuye Yang[1], Changyan Yi[1(✉)], and Junyi Wang[2]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
{shyou,mryyy,changyan.yi}@nuaa.edu.cn

[2] Key Lab of Cognitive Radio and Information Processing, Ministry of Education, Guilin University of Electronic Technology, Guilin, China
wangjy@guet.edu.cn

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 (ES). 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. Obviously, if such an issue cannot be well addressed, the reliability of microservices in serving corresponding tasks may be severely reduced. In this paper, 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 and Lagrangian method, which iteratively solves the problem over different timescales. Theoretical analyses and simulations evaluate the performance of the proposed solution, and show its superiority over counterparts.

Keywords: Edge computing · microservice deployment · layer sharing · reliability enhancement · online optimization

1 Introduction

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 [1,2]. Via this approach, multiple containers can share the machine's operating system

Z. Cai et al. (Eds.): WASA 2024, LNCS 14998, pp. 502–515, 2025.
https://doi.org/10.1007/978-3-031-71467-2_40

(OS) kernel and thereby do not require an OS per container, driving higher efficiencies, suitable for resource-limited edge nodes.

It is worth noting that 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 [3], 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 [1]. 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) [3] or across different ESs via distributed file systems (i.e., inter-ESs sharing) [4]. 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.

A key factor which should to be particularly considered in layer placement and loading problems is that, in practice, each ES may suffer from a variety of runtime failures caused by potential hardware breakdowns and configuration errors [5]. 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. 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 tasks collapse, but also the other ESs' mircoservcies loading common layers from it for serving all other tasks collapse.

How to enhance the system-wide reliability with aforementioned features is of great importance, while has not yet been studied in the literature. Technically, such an issue is challenged by the following aspects: *i*) 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 to be obtained [6,7]. *ii*) For each ES, frequent layer placements may cause the storage space fragmentation [8], 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 [3].

To fill the gap of the literature, in this paper, we design a novel two-timescale online management framework for edge computing with container-based microservice provisioning, including the optimization of *i*) placing which layer on each ES in the large timescale, and *ii*) 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 a two-timescale online algorithm based on modified Lyapunov optimization and stochastic rounding method to decouple decisions into different timescales and solve them separately. Theoretical analyses show that the proposed solution can well address the original problem with a low computational complexity.

The main contributions are summarized in the following.

- 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.
- Taking an equivalent problem reformulation, we propose an online algorithm based on the improved Lyapunov optimization method, which decomposes the long-term problem into a series of deterministic ones. Then, we decouple decisions into two different timescales, and develop an iterative algorithm to reach a near-optimum.
- Extensive numerical simulations are conducted to examine the feasibility of the proposed solution, and demonstrate its superiority compared to the counterparts.

The rest of this paper is organized as follows. In Sect. 2, we introduce the system model and problem formulation. Section 3 presents the detail of the proposed solution along with comprehensive analyses. Simulation results are given in Sect. 4, followed by conclusions in Sect. 5.

2 System Model

Consider an edge computing system 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 $\mathcal{S} = \{1, 2, \cdots, S\}$, 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 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.

Note that, layer placement may not be able to vary frequently in real-time because of the storage space fragmentation [8]. 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. 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 t-th time frame, and the index of the τ-th time slot in the t-th time frame as $\tau \in \mathcal{T}_t = \{tK, tK+1, \ldots, tK+K-1\}$.

2.1 Task Scheduling and Computation

Container-based microservices are often managed by applying container orchestration tool (e.g., Kubernetes [1]), which can adjust the microservice menu of ESs based on the requests of tasks, facilitating a better organization of layer placement and loading [3,9]. Since the preparation of such miroservice menus may be complicated and costly for ESs, 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

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

Here, we do not rule out the possibility that $m = m'$, i.e.,

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

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 $D_{m',m}^{s,tra}(\tau) = \frac{x_{m',m}^s(\tau)v_s}{r_{m',m}^{tra}(\tau)}$, where $r_{m',m}^{tra}(\tau)$ 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 $D_{m,s}^{sch}(\tau) = max\{D_{m',m}^{s,tra}(\tau)\}, \forall m' \in \mathcal{M}$.

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:

$$\sum\nolimits_{s\in\mathcal{S}} \rho_m^s(\tau) \leq 1, \forall s \in \mathcal{S}. \tag{3}$$

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 $D_{m,s}^{com}(\tau) = \frac{[\alpha_m^s(\tau)+\sum_{m'\in\mathcal{M}}(x_{m',m}^s(\tau)-x_{m,m'}^s(\tau))]v_s\beta_s}{\rho_m^s(\tau)f_m^{es}}$, where f_m^{es} represents CPU computation speed (measured by cycles/s) of each ES $m \in \mathcal{M}$, β_s is the number of CPU cycles required to complete each bit of these tasks.

2.2 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

$$\sum\nolimits_{l\in\mathcal{L}} d_m^l(t)v_l \leq \Omega_m^{ca}, \forall m \in \mathcal{M}, \tag{4}$$

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 $D_m^{pla}(t)$ can be calculated as $D_m^{pla}(t) = \sum_{l\in\mathcal{L}} d_m^l(t)v_l/r_{m,c}^{pla}(t)$, where $r_{m,c}^{pla}(t)$ 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.,

$$\vartheta_{m,m'}^l(\tau) \leq d_{m'}^l(t), \forall l \in \mathcal{L}, \forall m, m' \in \mathcal{M}. \tag{5}$$

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.,

$$\sum\nolimits_{m'\in\mathcal{M}} \vartheta_{m,m'}^l(\tau) \leq 1, \forall l \in \mathcal{L}, \forall m, m' \in \mathcal{M}. \tag{6}$$

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., $D_m^{loa}(\tau) = max\{\frac{\vartheta_{m,m'}^l(\tau)v_l}{r_{m,m'}^{loa}(t)}\}, \forall l \in \mathcal{L}, \forall m' \in \mathcal{M}$, where $r_{m,m'}^{loa}(t)$ denotes the loading rate between ES m and ES m' on a wired communication link.

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,

$$\lambda_m^s(t)\varpi_s^l \leq d_m^l(t) + \sum\nolimits_{m'\in\mathcal{M}} \vartheta_{m,m'}^l(\tau), \tag{7}$$

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 $D_{m,s}^{tol}(t) = D_m^{pla}(t) + \sum_{\tau \in \mathcal{T}_t} D_{m,s}^{exe}(\tau)$, where $D_m^{pla}(t)$ is the layer placement delay in time frame t, and $D_{m,s}^{exe}(\tau)$ is the edge execution delay for executing tasks requesting microservice s in time slot τ, i.e., $D_{m,s}^{exe}(\tau) = D_m^{loa}(\tau) + D_{m,s}^{com}(\tau) + D_{m,s}^{sch}(\tau)$, consisting of layer loading delay $D_m^{loa}(\tau)$, edge processing delay $D_{m,s}^{com}(\tau)$ and the delay to collect all tasks requesting miroservice s, i.e., $D_{m,s}^{sch}(\tau)$.

2.3 Problem Formulation

The lifetime of an ES refers to the time-length from the point of starting the operation to that of occurring malfunctions. Following the conventions in the literature [5], in each time slot τ, the lifetime $D_m^{life}(\tau)$ of ES m is defined to be an exponentially distributed random variable with parameter $c_m(\tau) \in [c_m^{min}, c_m^{max}]$, meaning that $Pr(D_m^{life}(\tau) \leq \gamma) = 1 - e^{-c_m(\tau)\gamma}$, where γ is the length of each time slot.

Due to the layer sharing, 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 $D_{m,s}^{upt}(\tau) = min\{D_{m'}^{life}(\tau)\}, m' \in \mathcal{M}'(\tau)$, where $\mathcal{M}'(\tau) = \{m' \mid \vartheta_{m,m'}^l(\tau) = 1, m' \in \mathcal{M}, l \in \mathcal{L}\}$ 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 $D_{m,s}^{exe}(\tau)$ exceeds $D_{m,s}^{upt}(\tau)$, 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 $F_{m,s}^{fail}(D_{m,s}^{exe}(\tau))$, can be calculated as

$$\begin{aligned} F_{m,s}^{fail}(D_{m,s}^{exe}(\tau)) &= Pr\{D_{m,s}^{upt}(\tau) \leq D_{m,s}^{exe}(\tau)\} \\ &= 1 - \prod\nolimits_{m' \in \mathcal{M}'(\tau)} Pr\{D_{m'}^{life}(\tau) > D_{m,s}^{exe}(\tau)\} \\ &= 1 - e^{-\sum_{m' \in \mathcal{M}'(\tau)} c_{m'}(\tau) D_{m,s}^{exe}(\tau)}. \end{aligned} \tag{8}$$

Then, the reliability in serving tasks requesting microservice s at ES m in time slot $\tau \in \mathcal{T}_t$ can be computed by $REL_m^s(\tau) = 1 - F_{m,s}^{fail}(D_{m,s}^{exe}(\tau)), \forall s \in \mathcal{S}, \forall m \in \mathcal{M}$. 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

$$\mathcal{R} = \frac{1}{TK} \sum\nolimits_{\tau=0}^{KT-1} \sum\nolimits_{m \in \mathcal{M}} \sum\nolimits_{s \in \mathcal{S}} REL_m^s(\tau). \tag{9}$$

For maximizing $\mathcal{R}$, we are required to optimize i) microservice manu decision in any time frame t, ii) layer placement decision for each ES m in any time frame t, iii) layer loading decision for each ES m in any time slot $\tau \in \mathcal{T}_t$, iv) task scheduling decision in any time slot $\tau \in \mathcal{T}_t$, and v) the computing resource allocation in any time slot $\tau \in \mathcal{T}_t$, denoted in short by $\mathcal{J}_A(t) = \{\lambda_m^s(t), d_m^l(t)\}, \mathcal{J}_B(\tau) = \{\vartheta_{m,m'}^l(\tau), \rho_m^s(\tau), x_{m,m'}^s(\tau)\}$, which can be formulated as

$$\begin{aligned} \mathcal{P}_0: \quad & \max_{\mathcal{J}_A(t), \mathcal{J}_B(\tau)} \lim_{T\to\infty} \mathcal{R} \\ \text{s.t. } & (1), (2), (3), (4) - (7) \\ & \lim_{T\to\infty} \frac{1}{T} \sum\nolimits_{t=0}^{T-1} D_{m,s}^{tol}(t) \le D_{m,s}^{th}, \end{aligned} \tag{10}$$

where besides the aforementioned constraints (1), (2), (3), (4)–(7), (10) is the long-term average delay constraint, in which $D_{m,s}^{th}$ denotes a pre-determined delay threshold of all tasks requesting microservice s.

Solving problem $\mathcal{P}_0$ is very challenging because constraints (2) and (3) indicate the inclusion of discrete decision variables, while constraints (4)–(7) 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 optimization method [6,10] is no longer applicable. To address this issue, we develop a two-timescale online optimization algorithm for joint microservice deployment, layer placement and loading, computing resource management and task scheduling (called OMLRC). Specifically, we first decompose the long-term stochastic problem into a series of deterministic instant problems, each of which is further decoupled into two subproblems in different timescales. Then, we propose an iterative algorithm integrating randomized rounding and Lagrangian method to solve these subproblems.

3 A Two-Timescale Online Algorithm

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 $D_{m,s}^{tol}(\tau) = D_m^{pev}(\tau) + D_{m,s}^{exe}(\tau)$, where $D_m^{pev}(\tau) = \frac{D_m^{pla}(t)}{K}$ represents the layer placement delay in each time frame t evenly distributed into all $|\mathcal{T}_t| = K$ time slots, respectively. Therefore, we have

$$\begin{aligned} \mathcal{P}_1: \quad & \min_{\mathcal{J}_A(t), \mathcal{J}_B(\tau)} \lim_{T\to\infty} \mathcal{F} \\ \text{s.t. } & (1), (2), (3), (4) - (7) \\ & \lim_{T\to\infty} \frac{1}{TK} \sum\nolimits_{\tau=0}^{TK-1} D_{m,s}^{tol}(\tau) \le \frac{D_s^{th}}{K}, \end{aligned} \tag{11}$$

where $\mathcal{F} = \frac{1}{TK}\sum_{\tau=0}^{KT-1}\sum_{i=1}^{I} F_{m,s}^{fail}(D_{m,s}^{exe}(\tau))$. 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.

3.1 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

$$Q_m^s(\tau+1) = [D_{m,s}^{tol}(\tau) - D_s^{th}/K]^+ + Q_m^s(\tau), \tag{12}$$

with initial state $Q_m^s(0) = 0$. Then, we introduce the quadratic Lyapunov function: $L(\boldsymbol{\Theta}(\tau)) \triangleq \frac{1}{2}[\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} Q_m^s(\tau)^2]$, where $\boldsymbol{\Theta}(\tau) = [Q_1^1(\tau), \ldots, Q_M^S(\tau)]$. 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 [11,12], the conditional Lyapunov drift can be written as $\Delta(\boldsymbol{\Theta}(\tau)) = \mathbb{E}[L(\boldsymbol{\Theta}(\tau+K)) - L(\boldsymbol{\Theta}(\tau)) \mid \boldsymbol{\Theta}(\tau)]$, which measures the difference of the Lyapunov function between K consecutive time slots. Intuitively, by minimizing the Lyapunov drift, 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 $\Delta_V(\boldsymbol{\Theta}(\tau)) = \Delta(\boldsymbol{\Theta}(\tau)) + V \cdot \mathbb{E}[\mathcal{F} \mid \boldsymbol{\Theta}(\tau)]$, 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 bounded under any possible decisions in any time slot τ can be expressed as*

$$\Delta_V \leq B + \sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{m\in\mathcal{M}} \mathbb{E}\{Q_m^s(\tau)[D_{m,s}^{tol}(\tau)(\boldsymbol{\Theta}(\tau)) - D_s^{th}/K]|\boldsymbol{\Theta}(\tau)\} + V \cdot \mathbb{E}[\mathcal{F}], \tag{13}$$

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

Proof. This proof is omitted due to the page limit.

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 $\Delta_V(\boldsymbol{\Theta}(t)) \leq BK + \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}}\sum_{\tau\in\mathcal{T}_t} \mathbb{E}\{Q_m^s(\tau)[D_{m,s}^{tol}(\tau) - D_s^{th}/K]|\boldsymbol{\Theta}(\tau)\} + V \cdot \sum_{\tau\in\mathcal{T}_t}\mathbb{E}[\mathcal{F}]$. Following the convention of the Lyapunov optimization method, the long-term stochastic optimization problem $\mathcal{P}_1$ can be decomposed into a series of deter-

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 (4) and (7);
for $s \in \mathcal{S}$ **do**
  Set $\widehat{\lambda}_m^s(t) = 1$ with the probability $\tilde{\lambda}_m^s(t)$;
end
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 $\delta_i(t)$;
  end
  else
    Set $\widehat{d}_m^l(t) = 1$ with the probability $\delta_i'(t)$;
  end
end
Output: Solution of $\mathcal{P}_t$: $\widehat{\lambda}_m^s(t)$ and $\widehat{d}_m^l(t)$.

ministic instant problem $\mathcal{P}_2$, which is given by

$$\begin{aligned}\mathcal{P}_2: \quad & \min_{\mathcal{J}_A(t),\mathcal{J}_B(\tau)} \mathcal{G}_{m,s}(t) \\ & = \sum_{m\in\mathcal{M}} \sum_{s\in\mathcal{S}} \sum_{\tau\in\mathcal{T}_t} \{Q_m^s(t)[D_{m,s}^{tol}(\tau) - D_s^{th}/K]\} + V \cdot \sum\nolimits_{\tau\in\mathcal{T}_t} [\mathcal{F}], \\ & \text{s.t. } (1),(2),(3),(4)-(7).\end{aligned}$$

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.

3.2 Algorithm Design for Large-Timescale Decisions

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

$$\begin{aligned}\mathcal{P}_t: \quad & \min_{\lambda_m^s(t),d_m^l(t)} \sum_{m\in\mathcal{M}} \sum_{s\in\mathcal{S}} Q_m^s(t)[D_m^{pla}(t) + D_m^{loa}(\tau)] + V \cdot \mathcal{F} \\ & \text{s.t. } (1),(4),(7).\end{aligned}$$

In order to solve this integer programming problem $\mathcal{P}_4$, we adopt randomized rounding technique [13] 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 $\lambda_m^s(t) \in \{0,1\} \rightarrow \lambda_m^s(t) \in [0,1]$, $d_m^l(t) \in \{0,1\} \rightarrow d_m^l(t) \in [0,1]$. Then, $\mathcal{P}_t$ can be solved in a polynomial time by using the linear programming solver, 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

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

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

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 $\widehat{\mathcal{P}}_t - \mathcal{P}_t^* \leq \Lambda$, where $\Lambda = \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(t) \{\sum_{l \in \mathcal{L}} \sum_{m \in \mathcal{M}} \frac{v_l}{r_{m,c}^{pla}(t)} + D_m^{loa}(\tau)\} + V$.*

Proof. This proof is omitted due to the page limit.

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

$$\begin{aligned} \mathcal{P}_{tK}^{'} : & \min_{\vartheta_{m,m'}^l(tK)} \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} Q_m^s(tK) D_m^{loa}(tK) + V\mathcal{F} \\ & \text{s.t. } (5) - (7). \end{aligned} \tag{14}$$

By relaxing the integer decision variable $\vartheta_{m,m'}^l(tK) \in [0,1]$, problem $\mathcal{P}_{tK}^{'}$ becomes a linear program (LP) problem, which can be solved by Lagrangian method. Finally, set the obtained solution $\widehat{\vartheta}_{m,m'}^l(tK)$ to $\{0,1\}$ by a rounding manner.

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:

$$\mathcal{P}_{tK}^{''}: \min_{x_{m,m'}^{s}(\tau),\rho_{m}^{s}(\tau)} \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}} Q_{m}^{s}(tK)[D_{m,s}^{com}(tK)+D_{m,s}^{sch}(tK)]+V\mathcal{F} \quad \text{s.t. } (1),(2),(3). \tag{15}$$

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

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.

3.3 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

$$\mathcal{P}_{\tau\neq tK}: \min_{\mathcal{J}_{B}(\tau\neq tK)} \sum_{m\in\mathcal{M}}\sum_{s\in\mathcal{S}} Q_{m}^{s}(tK)D_{i}^{exe}(\tau)+V\cdot\mathcal{F} \quad \text{s.t. } (1),(2),(3),(5)-(7). \tag{16}$$

Algorithm 2: Iterative Algorithm OMLRC

Initialize: Observe the instantaneous queue set $\boldsymbol{\Theta}(t)$, task generation $\alpha_{m}^{s}(\tau)$ and ESs' lifetime $D_{m}^{life}(\tau)$; **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;
      end
      Update the value of delay overflow queue $Q_{m}^{s}(t)$;
    end
    Update the iteration index $j = j + 1$;
  end
end
Output: $\lambda_{m}^{s}(t)^{\dagger}$, $d_{m}^{l}(t)^{\dagger}$, $\vartheta_{m,m'}^{l}(tK)^{\dagger}$, $x_{m,m'}^{s}(tK)^{\dagger}$, $\rho_{m}^{s}(tK)^{\dagger}$.

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 $\mathcal{P}_{tK}^{''}$.

4 Simulation Results

Table 1. Simulation Parameters

Param	Value	Param	Value	Param	Value
K	10	M	10	v_s	[0.5, 2] Mb
$r^{pla}_{m,c}(t)$	5 Mb/s	Ω^{ca}_m	50 Gb	$C_m(\tau)$	[0.01, 0.5]
$\alpha^s_m(\tau)$	[20, 50]	$r^{loa}_{m,m'}(t)$	[10, 20] Mb/s	f^{es}_m	20 GHz
v_l	[1, 3] Gb	β_s	300 cycles/bit	S	10

Simulations are conducted to numerically evaluate the performance of the proposed OMLRC. Table 1 lists the values of main simulation parameters. Furthermore, to show the superiority of the proposed OMLRC, the performances of the following schemes are also evaluated as benchmarks: *i*) JMDLS [3]: 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. *ii*) O2TL [14]: 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. *iii*) EGO [15]: The dynamic task generations is 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.

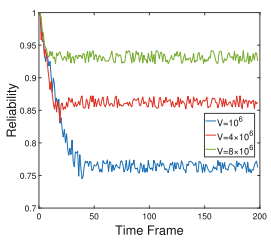

Fig. 1. The convergence of solving $\mathcal{P}_0$.

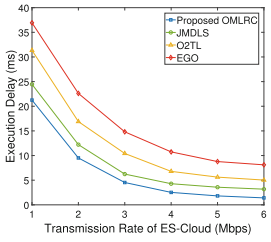

Fig. 2. Execution delay w.r.t $r^{pla}_{m,c}(\tau)$.

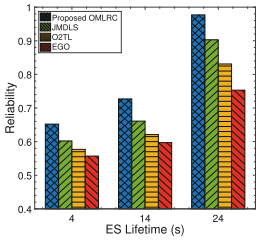

Fig. 3. Reliability w.r.t ESs' lifetime.

In Fig. 1, 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. In addition, when $V = 8 \times 10^6$, OMLRC converges faster than the other benchmark schemes. This can be attributed to the decrease

in V, which indicates a higher utilization rate of edge resources in the edge computing system, resulting in reduced execution delay and improved reliability. Consequently, it becomes necessary to perform multiple iterations between layer placement and layer loading in order to achieve optimal decisions.

Figure 2 shows the performance comparison of execution delay with respect to the transmission rate of cloud-ES. 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 $D_m^{pla}(t)$. Furthermore, this figure demonstrates that EGO and O2TL have limited control over 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, OMLRC effectively reduces execution delay by employing a two-timescale control mechanism for layer placement and layer loading.

Figure 3 shows the performance comparison of reliability with respect to ESs' lifetime. In this figure, as ESs' lifetime increases, the reliability also increases correspondingly. This trend can be attributed to the fact that with a longer ESs' lifetime, tasks requesting microservices can be more easily executed within the ESs' lifetime range. Additionally, ESs tend to prioritize layer loading over layer placement to avoid the placement delay, thereby enhancing reliability. 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. In contrast, 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, in an online manner.

5 Conclusion

This paper 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, layer loading and task scheduling across multiple ESs. Extensive simulations demonstrate that our proposed solution exhibits remarkable enhancements.

Acknowledgments. This work was supported by the State Key Laboratory of Massive Personalized Customization System and Technology under grant No. H&C-MPC-2023-04-01, National Natural Science Foundation of China (NSFC) under grant No. 62176122, Postgraduate Research & Practice Innovation Program of NUAA under grant No. xcxjh20231601, and by Postgraduate Research & Practice Innovation Program of Jiangsu Province under grants No. KYCX22_0372.

References

1. Tang, Z., Lou, J., Jia, W.: Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing. IEEE Trans. Mobile Comput. (2022)
2. Shi, Y., Yang, Y., Yi, C., Chen, B., Cai, J.: Towards online reliability-enhanced microservice deployment with layer sharing in edge computing. IEEE Internet Things J. 1 (2024)
3. Gu, L., Chen, Z., Xu, H., Zeng, D., Li, B., Jin, H.: Layer-aware collaborative microservice deployment toward maximal edge throughput. In: Proceedings of IEEE INFOCOM, pp. 71–79. IEEE (2022)
4. Zheng, C., et al.: Wharf: sharing docker images in a distributed file system. In: Proceedings of ACM SOCC, pp. 174–185 (2018)
5. Liu, J., Zhou, A., et al.: Reliability-enhanced task offloading in mobile edge computing environments. IEEE Internet Things J. **9**(13), 10:382–10:396 (2021)
6. Zhou, R., Wu, X., Tan, H., Zhang, R.: Two time-scale joint service caching and task offloading for UAV-assisted mobile edge computing. In: Proceedings of IEEE INFOCOM, pp. 1189–1198. IEEE (2022)
7. Shi, Y., Yi, C., Wang, R., Wu, Q., Chen, B., Cai, J.: Service migration or task rerouting: a two-timescale online resource optimization for MEC. IEEE Trans. Wirel. Commun. (2023)
8. Qiu, H., Noura, H., Qiu, M., Ming, Z., Memmi, G.: A user-centric data protection method for cloud storage based on invertible DWT. IEEE Trans. Cloud Comput. **9**(4), 1293–1304 (2019)
9. Yuan, Y., Yi, C., Chen, B., Shi, Y., Cai, J.: A computation offloading game for jointly managing local pre-processing time-length and priority selection in edge computing. IEEE Trans. Veh. Technol. **71**(9), 9868–9883 (2022)
10. Shi, Y., Yi, C., Chen, B., Yang, C., Zhu, K., Cai, J.: Joint online optimization of data sampling rate and preprocessing mode for edge-cloud collaboration enabled industrial IoT. IEEE Internet Things J. **9**(17), 16:402–16:417 (2022)
11. Georgiadis, L., Neely, M.J., Tassiulas, L., et al.: Resource allocation and cross-layer control in wireless networks. Found. Trends Netw. **1**(1), 1–144 (2006)
12. Yang, Y., et al.: Dynamic human digital twin deployment at the edge for task execution: a two-timescale accuracy-aware online optimization. arXiv preprint arXiv:2401.16710 (2024)
13. Srinivasan, A.: Approximation algorithms via randomized rounding: a survey. In: Proceedings of Advanced Topics in Mathematics (PWN), pp. 9–71 (1999)
14. Li, X., Zhang, X., Huang, T.: Asynchronous online service placement and task offloading for mobile edge computing. In: Proceedings of IEEE SECON, pp. 1–9 (2021)
15. Cao, K., Cui, Y., et al.: Edge intelligent joint optimization for lifetime and latency in large-scale cyber–physical systems. IEEE Internet Things J. **9**(22), 22:267–22:279 (2021)