Received: 29 January 2019 | Revised: 26 April 2019 | Accepted: 11 June 2019
DOI: 10.1002/cpe.5436

WILEY

SPECIAL ISSUE PAPER

Dynamic service deployment for budget-constrained mobile edge computing

Jingya Zhou[1,2,3] | Jianxi Fan[1] | Jin Wang[1] | Juncheng Jia[1]

[1]School of Computer Science and Technology, Soochow University, Suzhou, China
[2]College of Computing, Georgia Institute of Technology, Atlanta
[3]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Correspondence
Jingya Zhou, School of Computer Science and Technology, Soochow University, Suzhou 215006, China.
Email: jy_zhou@suda.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 61502328, 61572337, 61672370 and 61702351; Opening Foundation of Provincial Key Laboratory for Computer Information Processing Technology, Grant/Award Number: KJS1740; Postdoctoral Research Funds, Grant/Award Number: 1701173B; Jiangsu Overseas Visiting Scholar Program for University Prominent Yong & Middle-aged Teachers and Presidents

Summary
Currently, Mobile edge computing (MEC) is facing a great challenge that is how to make full use of edge resources to provide a seamless support for compute-intensive latency-sensitive applications. Prior studies often make a simple assumption that tasks can be executed upon every edge server, but the assumption does not hold in practical scenarios. Because a specific application task often corresponds to a certain service that provides the corresponding running environment, whereas an edge server only has limited resources and cannot offer too many services. How to decide service deployment of so many types of services among multiple edge servers is also a big challenge. To address the challenge, we study dynamic service deployment for latency-sensitive applications. We first model the long-term budget-constrained latency minimization problem as a multi-slot latency minimization problem based on the Lyapunov framework. By doing this, the hardness of a problem is significantly reduced, since we never require future information to solve the long-term optimization. Furthermore, we extend our study by joining the task scheduling optimization, where every edge server is fully utilized in an even more efficient collaborative manner. Our extensive experiments show that the proposed algorithms can bring short latency with low cost.

KEYWORDS
latency-sensitive applications, Lyapunov optimization, mobile edge computing, service deployment, task scheduling

1 | INTRODUCTION

Nowadays, mobile devices (eg, smartphones, wearable devices, etc) are becoming enormously popular along with the rapid development of the Internet of Things (IoT).[1] The next generation cellular communication technology provides pervasive network coverage, which drives many mobile applications, such as interactive online gaming, face recognition, and virtual/augmented reality (VR/AR). Those applications are commonly compute-intensive and latency-sensitive, and are deemed to be killer applications for modern mobile devices due to their limited resources. Cloud computing can be used to handle the computation tasks, but applications' strict requirements on low latency are very hard to meet when communications traverse through the Internet backbone.

Recently, edge (fog) computing[2-4] is becoming more mentioned, especially for Mobile edge computing (MEC),[5-8] which has been acknowledged as a fascinating technology trend for future computing paradigm. Compared to running application tasks on mobile devices or powerful servers located in a remote cloud, MEC allows end users to send application tasks and corresponding data via wireless (or cellular) networks to edge servers that are geographically adjacent to users. Thereby, MEC can implement ultra-low response latency with high confidence and is especially suitable for those compute-intensive yet latency-sensitive applications. The current business mode has been changed with the rapid development of MEC techniques, where a new role, ie, edge service provider, has emerged between cloud providers and telecom operators. By considering the resource limitation in MEC environments, a considerable amount of existing works[9-16] focus on the major challenge on task offloading. No matter whether from an end user's perspective or an edge service provider's perspective, most of these efforts share a common assumption that tasks sent from end users can be executed upon every edge server. However, this assumption neglects the fact that a specific application task often requires installing a certain service to provide the corresponding running environment. For example, when you play a VR game outside on

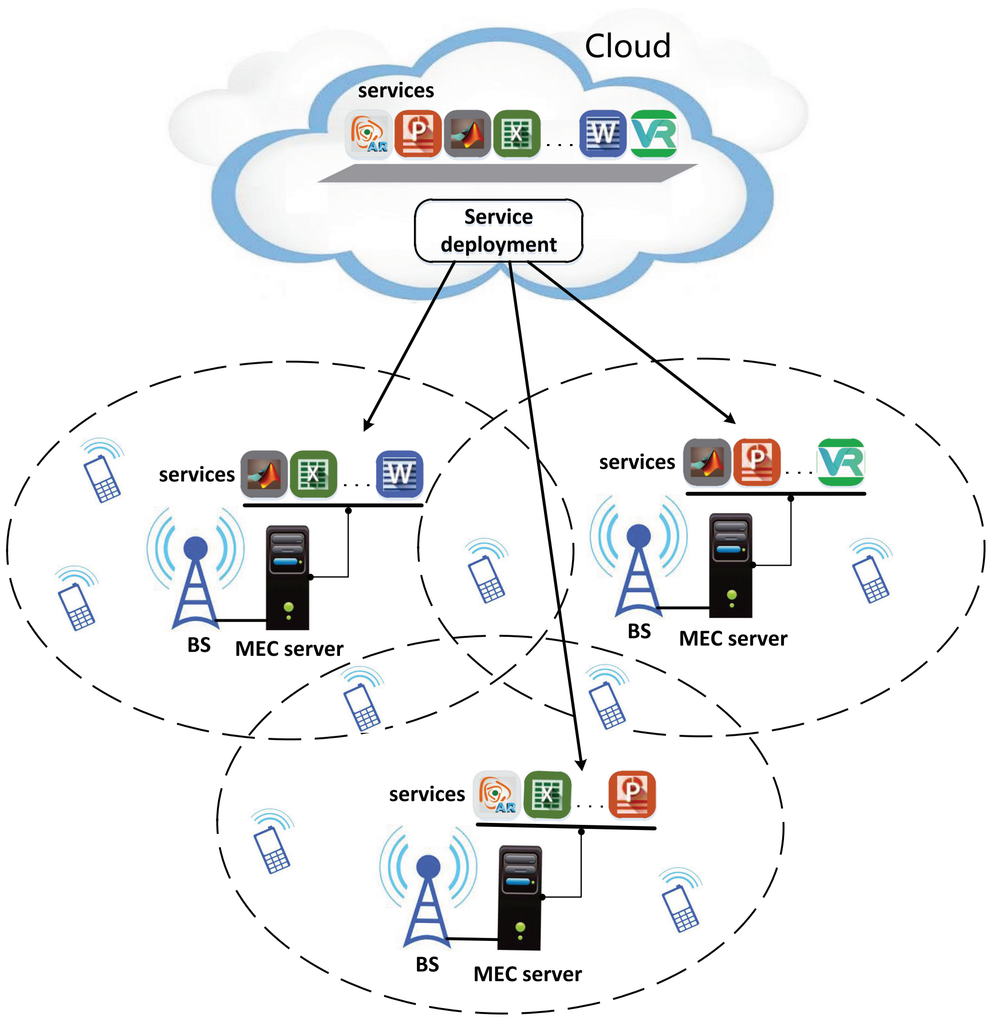

FIGURE 1 An example scenario to illustrate the service deployment problem for latency sensitive applications in the MEC environment

your own smartphone, to enhance user experience of fluency, partial application tasks can be offloaded to the edge server as long as it installs the corresponding VR service.

In this paper, we primarily study the dynamic service deployment problem from an edge service provider's perspective. The MEC system is shown in Figure 1, where many base stations (BS) construct a large scale radio access network, and each of them is equipped with an edge server (ie, MEC server) for the purpose of offering end users nearby computing power and storage capability. A cloud hosts all services; the edge service provider deploys services from the cloud to its MEC servers so as to support extensive applications. Each BS provides the radio access within a limited distance, and the areas covered by different BS might overlap one another. End users inside the overlapping area can choose an arbitrary BS that covers them to send requests (eg, choose the BS with the strongest signal). When we talk service deployment, we mean installing the applications' programs on MEC servers, eg, install program codes, libraries, and databases. For the edge service provider, service deployment is of great challenge, which is reflected in the following aspects: First, owing to the resource limit of MEC server, only a small subset of services can be deployed on each server. It is hard to decide which services to deploy such that all arrived tasks can be met. Second, if an MEC server cannot handle the arrived tasks sent from end users, scheduling tasks to the cloud is probably not a feasible solution due to the violation of low-latency requirement. Then, the server has to carefully collaborate with other servers and share these tasks to others that meet the service requirements. Third, the MEC system is always in a dynamic environment in which end users' trajectories are usually stochastic. The application popularity (ie, service popularity) and the demands often change at any time. How to adaptively deploy services from the long-term perspective makes the problem more challenging.

We investigate the dynamic service deployment together with task scheduling in the MEC environment, and make significant contributions described as follows:

- We build a new MEC system model, and define three types of costs to identify the operation cost paid by an edge service provider. We present a theoretical method to analyze the impact of service deployment upon the latency and cost, and formulate the service deployment problem from an edge service provider's perspective.
- To capture the long-term feature of service deployment problem, we divide the original problem into multi-slot minimization sub-problems by leveraging the Lyapunov optimization idea, which significantly reduces the hardness of the original problem.

- We combine service deployment and task scheduling together to implement a joint optimization effect on response latency and operation cost. We also design online algorithms to efficiently address the proposed problems.
- We perform extensive experiments to prove the effectiveness and efficiency of our work.

The remainder of this paper is organized as follows. Section 2 introduces the related work and Section 3 details the MEC system model and defines the service deployment problem. Section 4 elaborates the design of our online algorithm. Section 5 further explores a joint optimization problem. In Section 6, we evaluate our proposed algorithms by conducting extensive experiments. Section 7 concludes this paper.

2 | RELATED WORK

Following the idea of offloading computation and data to the edge, many studies focus on task offloading and resource allocation in MEC environments. Chen et al[11] formulated a distributed offloading game to capture the offloading problem, where users are able to make a decision on whether offloading tasks and how much computation to be offloaded. They also implemented a distributed algorithm in dynamic scenarios. Considering an energy harvesting (EH) technique, Mao et al[12] focused on a green MEC system equipped with EH devices and proposed an efficient offloading mechanism. Tan et al[14] created a generalized model to study job dispatching and scheduling among multiple edge servers and designed a provable approximate algorithm. Wang et al[17] further took user mobility into account and formulated a dynamic resource allocation problem given that tasks are offloaded on edge servers. They introduced a gap-preserving transformation of the problem, and correspondingly developed an online algorithm that can provide a parameterized competitive ratio. Jia et al[18] concentrated on cost minimization caused by dynamic changes of offloading request, and proposed a prediction method to decide whether releasing or creating instances of network functions inside various cloudlets. Chen et al[19] considered an energy constrained scenario, and defined a peer offloading game among multiple BSs. Through analyzing the equilibrium and corresponding efficiency loss, they designed a strategy to guide every BS's task offloading decision, so as to enable the distributed and autonomous offloading by edge servers themselves.

These existing works have an implicit yet impractical assumption that tasks could be handled on an arbitrary server on matter whether the required services have been deployed. Similar to service deployment, data/content caching has been studied by many previous literatures. Shanmugam et al[20] investigated the content caching problem in small cell networks to minimize content access latency. Prabh and Abdelzaher[21] developed an application service to search the optimal locations for caching data and reducing packet transmission power. Müller et al[22] put forward a new approach to learn the knowledge regarding context-specific popularity, and then updated content cached on each BS adaptively. Different from data/content caching where the major concern is concentrated on the caching under server's storage limit, a feasible service deployment should take into account not only storage capacity but also computing power. Moreover, allowing task sharing among servers makes the problem more complicated.

Xie et al[23] studied service caching problem by allowing edge servers to collaboratively cache the services. The goal of Xie et al[23] is to minimize the traffic load forwarded to the remote cloud, but the response latency is not guaranteed. Xu et al[24] studied a service caching problem by taking into account both response latency and energy consumption. However, they assumed that part of arrived tasks could be shared to a cloud rather than other edge servers, and it is unreasonable especially for most of latency-sensitive applications since low latency is hard to be ensured. Besides, it only applies to separable tasks scenario. He et al[15] investigated the optimal provisioning of services among edge servers with sharable and non-sharable resources, and presented efficient algorithms based on maximum flow method to maximize the requests sent by end users without caring about response latency. Wang et al[25] studied service entity placement for VR applications in MEC environments. The service entity here refers to the bundle of user's individual data and processing logics, while in our study, different services correspond to different applications, so their study only works for one application scenario.

Our prior work[16] investigated the service deployment problem by taking both response latency and operation cost into account. However, we did not consider the transmission cost when updating service deployment in a new time slot. Moreover, the user's task requests are assumed to be scheduled by following a simple yet not efficient principle. Different from the single task scheduling,[26] the problem becomes more complicated when coupled with the uncertainty of service placement. Therefore, we further combine service deployment and task scheduling together in this paper to optimize the expected response latency as well as reduce cost.

3 | SYSTEM MODEL AND PROBLEM FORMULATION

We start by introducing the MEC system model, and discuss service deployment in the model from an edge service provider's point of view. After that, we analyze theoretically the response latency and multiple types of costs given the service deployment. At last, we formally define the service deployment problem. Table 1 summarizes the key symbols used throughout this paper.

3.1 | System model

The MEC system is a hybrid edge cloud system, where a cloud system is used to provide backup support based on its powerful computing power and storage capacity, and a radio access network provides pervasive access services with n BSs . The cloud usually is far away from end users,

TABLE 1 Key symbols used throughout this paper

Symbol	Description
n	The number of MEC servers
m	The number of service types
θ	The variable of computation time
S_x	The maximum set of services installed on server *x*
(e_r, p_r, π_r)	Service type vector, computation demand, and data size of task *r*
$d_{x,i,t}$	Binary decision variable denoting whether service *i* is installed on server *x* in time slot *t*
$\phi_{r,x,t}$	Offloading rate of task *r*
$I^{y}_{x,r,t}$	Task *r*'s scheduling decision in time slot *t*
$R'_{x,t}$	The set of tasks sent from end users on server *x*
$R^{x}_{y,t}$	The set of tasks scheduled from server *y* to *x* in time slot *t*
$R_{x,t}$	The set of tasks handled on server *x*
$R^{in}_{x,t}$	The set of tasks scheduled from other servers to server *x* in time slot *t*
$R^{local}_{x,t}$	The set of tasks handled locally on server *x* in time slot *t*
$T^{t}_{x-local}$	Average response latency of tasks handled on server *x* in time slot *t*
$T^{t}_{x,y}$	Time spent on task data transmission from server *x* to *y*
T^{t}_{x-y}	The average transmission time of tasks shared out from server *x* to server *y* in time slot *t*
T^{t}	The expected response latency in time slot *t*
$\lambda_{x,i,t}$	The arrival rate of type *i*'s tasks on server *x* in time slot *t*
$\lambda^{in}_{x,i,t}, \lambda^{out}_{x,i,t}$	The sharing in rate and sharing out rate of type *i*'s task on server *x* on time slot *t*
$\lambda'_{x,i,t}$	The actual arrival rate of type *i*'s tasks handled on server *x* in time slot *t*
$\lambda^{local}_{x,t}$	The arrival rate of tasks handled locally on server *x* in time slot *t*
$C^{t}_{inst}, C^{t}_{comp}, C^{t}_{traf}$	Cost paid for installation, computation, and communication traffic in time slot *t*
$\alpha_t, \beta_t, \varphi_t$	Price on service installation, computation, and transmission
C^{t}	Cost in time slot *t*
$Q(t)$	Queue backlog in time slot *t*
$\Psi(\cdot)$	Lyapunov function
$o(\cdot)$	Drift function

and it preserves a complete set of *m* services for all applications. Each BS is co-located with an MEC server (we do not distinguish obviously BS and MEC server in this paper). Services are deployed from the cloud to all MEC servers. Each BS can offer the radio access within a restricted area, and nowadays, the average area size in a 4G network is about 0.07-0.12 km^2.[27] The radio access network is dense enough to ensure that end users holding mobile devices are often covered by more than one BS, and thereby, they are able to send application tasks to any BS covering them. Considering that each MEC server has the limited computing and storage resources, the number of services allowed to install on a server is also limited. Here, we use S_x to represent the maximum set of services installed on server *x*.

In this paper, we assume a one-to-one mapping from task to service, which means a kind of tasks corresponds to a specific service. When a specific task arrives at server *x*, if the required service has been installed, the task will be handled locally on server *x*; otherwise, the task cannot be handled by server *x*. The required services are deployed on edge servers in advance. Here, the required services are varied depending on the specific applications running on end users' cell phones. In general, most of network-based applications are developed in client/server mode, where client software runs on end users' terminals while server software (or middleware) runs on the server. Tasks generated from client can be handled by the corresponding server software, and here, the server software just corresponds to the service entity. Therefore, our assumption holds for most of the applications. However, many existing studies[23,24] assume that those locally unhandled tasks are sent to cloud for execution. We argue that the assumption is not always held for most of today's latency-sensitive applications. When MEC servers communicate with cloud, they have to rely on the Internet backbone. The size of data to be processed in each task is usually not negligible, and transferring task data over the Internet backbone would easily cause unacceptable latency for end users. To avoid bad user experience on response latency, we explore making full use of MEC servers to handle the arrived tasks in a cooperative manner. In our model, when a server cannot handle a task, the server has to share the task to another server that matches the task's service requirement. MEC servers are usually connected via the wired network, and telecom operators indeed connect their BSs through high speed fiber network to provide high bandwidth and support low latency network services accordingly. The metropolitan area network actually reflects the above scenario. As a result, end users' task requests can be handled among MEC servers without external executions on a remote cloud in our model. We confirm that the service required by an arbitrary task has been installed in our system with high confidence as long as the system scale is large enough.

If the required service of a task has not been installed before, there are two methods to address the problem. First, the task is sent to the cloud for further execution; second, downloading the required service to the edge server. We can design an algorithm to estimate the latencies of two methods and conduct the method with lower latency. If the estimation accuracy is not acceptable, two methods conduct at the same time and then we can always ensure the latency as low as possible. The mismatching cases are very rare; thereby, the system performance will not be influenced significantly.

3.2 | Service deployment

In general, each application corresponds to a specific service, and we use a triple (e_r, p_r, π_r) to describe an application task *r*, where e_r refers to the required service type and is represented by an *m*-dimensional vector, for example, $e_r = (0, 1, \ldots, 0)$ implies that task *r*'s required service type is 2. p_r refers to the computation demand and is measured based on CPU cycles here. As for task *r*'s data to be processed, its data size is represented by π_r. The service types required by arrived application tasks are diverse, while each MEC server hosts only a small subset of services, and thereby service deployment should be studied so as to decide the specific set of services installed on every MEC server. A service deployment algorithm is assumed to be executed on the cloud and generate an overall service placement solution.

The deployment algorithm is an important component in the MEC system, and the edge service provider uses it to manage the services running in the system. The time in our system model is discrete and is divided into Γ time slots. As the tasks vary over time, the service deployment updates at each time slot accordingly. Note that the length of a time slot is assumed to match the timescale at which an update of service deployment can be finished.

Let $d_{x,i,t}$ denote the basic deployment decision, it is a binary variable and takes the value of 1 if service *i* is installed on server *x* in time slot *t*; otherwise, it takes the value of 0. For an arbitrary MEC server *x* concerned, its deployment decision is denoted by a vector composed of *m* basic elements, ie, $d_{x,t} = (d_{x,1,t}, d_{x,2,t}, \ldots d_{x,m,t})$. Due to a server's resource limit, the number of services installed on server *x* should not exceed $|S_x|$, ie, $\sum_{i=1}^{m} d_{x,i,t} \leq |S_x|$.

3.3 | Latency analysis

In the proposed system model, end user's tasks may be handled on the first arrived MEC server or be shared out to another MEC server. In this section, we analyze response latency by taking into account two scenarios.

3.3.1 | Local handling

Considering service matching, not all of tasks received from end users are handled locally on server *x*, while those unmatched tasks are shared to other MEC servers. In the first scenario, we only focus on the tasks handled locally, and the average response latency of tasks handled locally on server *x* in time slot *t*, $T^t_{x-local}(d_{x,t})$, generally consists of two components, the time spent on data offloading, T^t_{x-off}, and the time spent on task handling, $T^t_{x-handle}$. Combining them together, we get the latency

$$T^t_{x-local}(d_{x,t}) = T^t_{x-off} + T^t_{x-handle}(d_{x,t}). \tag{1}$$

Similar to prior work,[11,13] we neglect the time spent on the result transmission from server back to the end user. Our settings are based on the following considerations: Firstly, for most applications, the size of output result is much smaller than the size of task data to be processed. Therefore, the time spent on returning results is very small compared with time spent on sending tasks and handling. For example, when you play an online game, all your operation data will be sent to the server, and the server returns your current position and status back after a series of computations. Compared with operation data, the returning data are very small. Secondly, even if the returning data are not less than the sending data, it does not affect the effectiveness of our model because the time spent on returning data is independent from our model.

Let $R_{x,t}$ denote the set of tasks on server *x* sent from end users in time slot *t*, and $\phi_{r,x,t}$ denote the offloading rate of task *r*, $\forall r \in R_{x,t}$; then, we can calculate the offloading time by

$$T^t_{x-off} = \frac{1}{|R_{x,t}|} \sum_{r \in R_{x,t}} \frac{\pi_r}{\phi_{r,x,t}}. \tag{2}$$

In a radio access network environment, each BS is known to manage multiple wireless channels for communications including uplink and downlink channels, and the channel is usually assumed to be allocated evenly among tasks. Tasks are offloaded via the uplink channel, and channels are orthogonal so they do not interact with one another. In contrast, when transmissions happen simultaneously via the same channel, the offloading rate would decline due to interference. Let us assume that each BS has *k* uplink wireless channels, and then, the average number of tasks transmitting via the same channel in time slot *t* should be $|R_{x,t}|/k$. In terms of the conclusion reported in literature,[28] the offloading rate of an arbitrary task *r* in time slot *t* is calculated as follows:

$$\phi_{r,x,t} = w\log_2\left(1 + \frac{q_{r,t}g_{r,x,t}}{\varpi + \sum\limits_{i=1 \wedge i \neq r}^{|R_{x,t}|/k} q_{i,t}g_{i,x,t}}\right), \tag{3}$$

where *w* is the channel bandwidth, ϖ is the background noise power, $q_{r,t}$ is the transmission power, and $g_{r,x,t}$ is the channel gain between the end user who sends task *r* and the BS co-located with server *x*.

To further measure the task's local handling time, we leverage queuing theory to model each MEC server as an M/G/1 queue, where M implies that the task arrival is a Poisson process, while G implies that the computation time follows an arbitrary distribution. Thus, local handling time

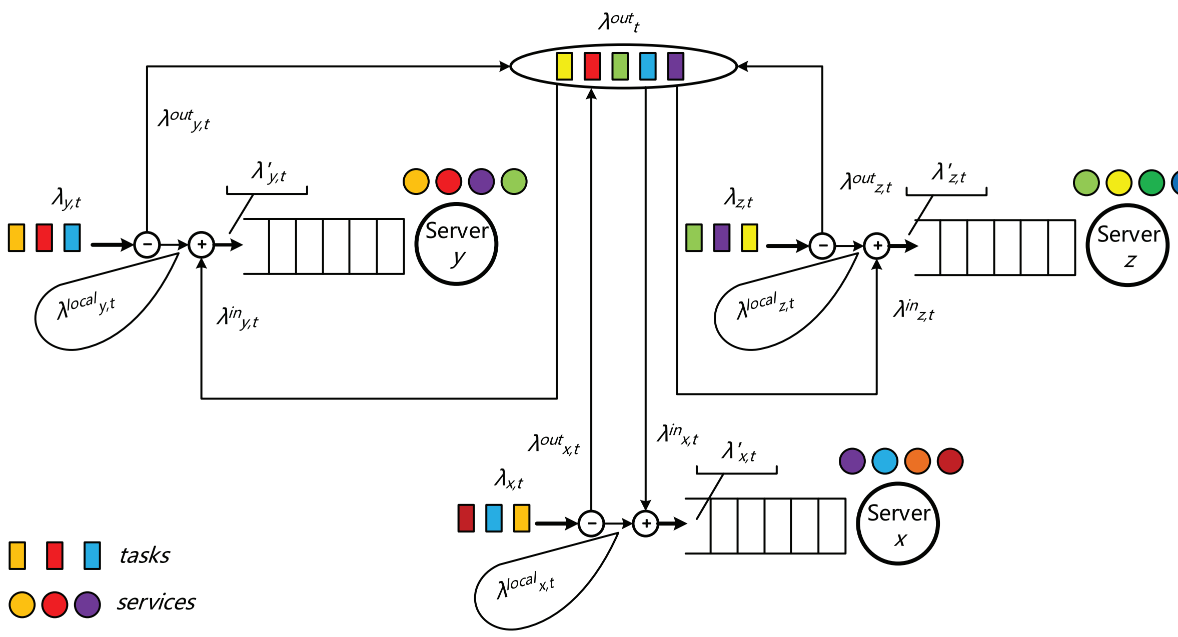

FIGURE 2 A simple example to illustrate the tasks sent from end users and shared among three MEC servers

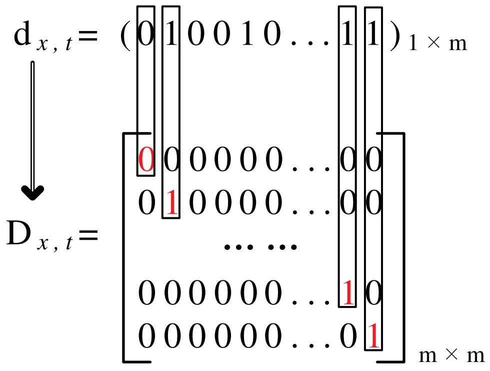

FIGURE 3 An illustration of how to generate a matrix $D_{x,t}$ from a vector $d_{x,t}$

$T^t_{x-handle}$ is the sojourn time of task in the queue, and consists of waiting time T^t_{x-wait} and computation time T^t_{x-comp}, ie,

$$T^t_{x-handle}(d_{x,t}) = T^t_{x-comp}(d_{x,t}) + T^t_{x-wait}(d_{x,t}). \tag{4}$$

Figure 2 shows an example of tasks sent from end users and shared among multiple servers, where $\lambda_{x,t}$ denotes the task arrival rate on server x in time slot t, $\lambda^{out}_{x,t}$ denotes the task sharing-out rate from server x in time slot t, and $\lambda^{local}_{x,t}$ denotes the arrival rate of tasks left on server x in time slot t. These arrival rates are represented by m-dimensional vectors, containing arrival rates of tasks corresponding to all service types. Tasks that can find matching services are left behind. We generate a $m \times m$ matrix $D_{x,t}$ from a decision vector $d_{x,t}$ in order to facilitate the calculation, and the generation method is illustrated in Figure 3, where each element $d_{x,i,t}$ of $d_{x,t}$ is placed on the position of the ith row and the ith column in matrix $D_{x,t}$, and other positions are filled with 0. Then, we obtain

$$\lambda^{local}_{x,t} = \lambda_{x,t} \cdot D_{x,t}, \tag{5}$$

$$\lambda^{out}_{x,t} = \lambda_{x,t} - \lambda^{local}_{x,t}. \tag{6}$$

We obtain the total task sharing-out rate λ^{out}_t by adding all of $\lambda^{out}_{x,t}$, ie,

$$\lambda^{out}_t = \sum_{x=1}^{n} \lambda^{out}_{x,t} = \left(\lambda^{out}_{1,t}, \lambda^{out}_{2,t}, \dots, \lambda^{out}_{m,t}\right)_{1\times m}. \tag{7}$$

λ^{out}_t is still a m-dimensional vector, where

$$\lambda^{out}_{i,t} = \sum_{x=1}^{n} \lambda^{out}_{x,i,t}. \tag{8}$$

After obtaining task sharing-out rate, in the following, we discuss task sharing-in rate, which is also derived from task sharing-out rates of other servers. If a task's service requirement cannot be matched locally, it would be shared out to another server that can match its requirement. There might be multiple candidate servers available, and then, the task is randomly shared out to any one of them. All deployment decision vectors make up a $n \times m$ decision matrix d_t. The number of non-zero values at the ith column in matrix d_t is denoted by num_i, which refers to how many servers have deployed service i. If a server x has deployed service i, it receives the sharing-in rate of tasks of type i that should be $\frac{\lambda_{i,t}^{out}}{num_i}$. We construct an $m \times m$ diagonal matrix with $\frac{1}{num_i}$, denoted by NUM_t. Then, the sharing-in task arrival rate on server x in time slot t is represented by

$$\lambda_{x,t}^{in} = \lambda_t^{out} \cdot NUM_t \cdot D_{x,t}. \tag{9}$$

Consequently, the actual arrival rate of tasks handled on server x in time slot t should be

$$\lambda'_{x,t} = \lambda_{x,t}^{local} + \lambda_{x,t}^{in}. \tag{10}$$

As a result, the set of tasks handled on server x is $|R'_{x,t}| = \sum_{i=1}^{m} \lambda'_{x,i,t} \cdot t$. As noted that task r's ($\forall r \in R'_{x,t}$) service requirement is represented by a vector e_r, if task r is handled on server x, we have $e_r \cdot d_{x,t}^T = 1$; otherwise, $e_r \cdot d_{x,t}^T = 0$, where $d_{x,t}^T$ is the transposition of vector $d_{x,t}$. Let f_x denote the computation power of server x (ie, represented by its CPU frequency); then, the average computation time on server x in time slot t is

$$T_{x-comp}^t(d_{x,t}) = \frac{1}{\left|R'_{x,t}\right|} \sum_{r \in R'_{x,t}} \frac{p_r}{f_x} \cdot e_r \cdot d_{x,t}. \tag{11}$$

Let θ denote the variable of computation time, and $\mathrm{E}[\theta]$, $\mathrm{E}[\theta^2]$ denote its first and second moments, respectively. According to Pollaczek-Khinchin mean formula,[29] the average waiting time on server x in time slot t is calculated by

$$T_{x-wait}^t(d_{x,t}) = \frac{\lambda'_{x,t} \cdot \mathrm{E}[\theta^2]}{2\left(1 - \lambda'_{x,t} \cdot \mathrm{E}[\theta]\right)}. \tag{12}$$

3.3.2 | Sharing out to another server

In this section, we focus on the scenario of sharing-out tasks. For these tasks, the average response latency $T_{x,y}^t(d_{x,t})$ consists of three components, ie, time spent on data offloading, T_{x-off}^t, task handling, $T_{x-y}^t(d_{x,t})$; and task data transmission, $T_{y-handle}^t(d_{x,t})$, respectively, ie,

$$T_{x,y}^t(d_{x,t}) = T_{x-off}^t + T_{y-handle}^t(d_{x,t}) + T_{x-y}^t(d_{x,t}). \tag{13}$$

The first two components are deduced in the same way as local handling scenario. For two arbitrary servers x and y, we already get the sharing-out rate of tasks of type i from server x in time slot t, $\lambda_{x,i,t}^{out}$ as shown in Section 3.3.1. In the random model, the sharing-out rate of tasks of type i from server x to y is $\frac{\lambda_{x,i,t}^{out}}{num_i}$ as long as server y has installed service i, ie, $d_{y,i,t} = 1$, and then, we obtain the number of tasks shared out from x to y in time slot t,

$$\left|R_{x,t}^y\right| = \sum_{i=1}^{m} \frac{\lambda_{x,i,t}^{out} d_{y,i,t}}{num_i}. \tag{14}$$

The average data size of tasks shared out from server x in time slot is

$$\pi_x^{out} = \frac{\sum_{r \in R_{x,t}} \left(1 - e_r \cdot d_{x,t}^T\right) \pi_r}{\sum_{r \in R_{x,t}} \left(1 - e_r \cdot d_{x,t}^T\right)}.$$

Let w_{xy} be the bandwidth from server x to y, and then, the average transmission time is calculated by

$$T_{x-y}^t(d_{x,t}) = \frac{\pi_x^{out}}{w_{xy}}. \tag{15}$$

At last, we obtain the expected latency of an arbitrary task in time slot t by combining two scenarios together,

$$T(d_t) = \frac{1}{n} \sum_{x=1}^{n} \left(T_{x-local}^t(d_{x,t}) \frac{\left|R_{x,t}^{local}\right|}{|R_{x,t}|} + \sum_{y \neq x}^{n} T_{x,y}^t(d_{x,t}) \frac{\left|R_{x,t}^y\right|}{|R_{x,t}|} \right). \tag{16}$$

3.4 | Cost model

We consider three aspects of costs regarding the management of an MEC system, including service installation cost, computation cost, and traffic cost. These costs are believed to represent the most prominent expenditure from the perspective of an edge service provider.

3.4.1 | Service installation cost

This cost corresponds to the expenditure paid for service copyright purchase and service maintenance, and it is charged in proportion to the number of services installed on servers. Let $\alpha_{i,t}$ be the installation price for service i in time slot t, where the service copyright purchase is charged at the period of authorization; thereby, it is assumed to be apportioned to multiple slots within the period. Specifically, the service installation cost on server x in time slot t is computed as

$$C_{inst}^{t}(d_{x,t}) = \sum_{i=1}^{m} \alpha_{i,t} d_{x,i,t}. \tag{17}$$

3.4.2 | Computation cost

This cost represents the expenditure paid for the task execution on the server. It is charged at the task's computation demand. As discussed in Section 3.3.1, the set of tasks handled on server x in time slot t is $R'_{x,t}$. Let β_t be the price of computation power in time slot t; the computation cost on server x in time slot t should be

$$C_{comp}^{t}(d_{x,t}) = \sum_{r \in R'_{x,t}} \beta_t p_r e_r \cdot d_{x,t}^{T}. \tag{18}$$

3.4.3 | Traffic cost

This cost represents the expenditure paid for data transmission including two components: one is the transmission of sharing task data when the task cannot be handled locally on the first arrived server (ie, outgoing traffic), the other is the transmission of service middlewares when service deployment updates (ie, incoming traffic). Both kinds of traffics are charged at the size of data transmitted. In order to facilitate the service deployment update, we define the following operation shown below:

$$\nabla \left(d_{x,i,t}, d_{x,i,t-1}\right) \triangleq \begin{cases} 1, & d_{x,i,t} = 1, d_{x,i,t-1} = 0, \\ 0, & \text{otherwise.} \end{cases}$$

As talked above, the set of tasks shared out from server x is $R_{x,t}^{out}$. Let π_i be the size of service i, and φ_t be the transmission price in time slot t; then, the traffic cost associated with x in time slot t is given by

$$C_{traf}^{t}(d_{x,t}) = \varphi_t \left(\sum_{r \in R_{x,t}^{out}} \left(1 - e_r d_{x,t}^{T}\right) \pi_r + \sum_{i=1}^{m} \nabla \left(d_{x,i,t}, d_{x,i,t-1}\right) \pi_i \right). \tag{19}$$

Consequently, the total cost in time slot t is the sum of above three aspects of costs, ie,

$$C(d_t) = \sum_{x=1}^{n} \left(C_{inst}^{t}(d_{x,t}) + C_{comp}^{t}(d_{x,t}) + C_{traf}^{t}(d_{x,t}) \right). \tag{20}$$

3.5 | Problem formulation

The purpose of this paper is to explore the optimization of MEC system by leveraging the service deployment from the perspective of an edge service provider. The problem takes end users' task requests and MEC servers as input. We notice that the actual requests cannot be obtained in advance. However, the time slot is short in our discrete time system, and the requests can be forecasted with a high accuracy.[30] We do not discuss the specific forecasting method and just assume task requests are given. Our goal is to continuously decide what services are placed on each MEC server such that the response latency is minimized while the total cost does not exceed the budget. The dynamic service deployment problem is formulated as follows.

Definition 1 (Dynamic service deployment). In an MEC system, there are n MEC servers and m services available for deployment. The time is divided into Γ time slots, and $T(d_t)$ and $C(d_t)$ denote the response latency and total cost respectively, where d_t denotes the service deployment matrix in time slot t, $\forall t \leq \Gamma$. The objective is to find d_t in every time slot t to minimize response latency under a given budget constraint B, ie,

$$\begin{aligned} &\arg\min \lim_{\Gamma \to \infty} \frac{1}{\Gamma} \sum_{t=1}^{\Gamma} T(d_t) \\ &s.t. \\ &(1) \lim_{\Gamma \to \infty} \frac{1}{\Gamma} \sum_{t=1}^{\Gamma} C(d_t) \leq B, \\ &(2) \sum_{i=1}^{m} d_{x,i,t} \leq |S_x|, \forall x \in [1, 2, \ldots, n]. \end{aligned}$$

4 | ONLINE ALGORITHM DESIGN

The solution of service deployment problem must realize the optimization from a long-term perspective. In this section, we firstly model the long-term service deployment problem as the multi-slot minimization problem by leveraging a Lyapunov optimization framework. Afterwards, we design an online algorithm and show its effectiveness.

4.1 | Multi-slot minimization problem

As time elapses (ie, $\Gamma \to \infty$), the major challenge of our proposed service deployment problem is the long-term cost budget coupling service deployment matrix across multiple time slots. Lyapunov optimization framework is a widely used tool to analyze the stability of solutions for a dynamic system. Thereby, we decide to solve the problem by leveraging a Lyapunov optimization mechanism.

To meet the long-term budget constraint, we create a virtual queue Q. We use the queue to record the cost deficits in all time slots from the beginning to the present, and then, let the queue to provide the guidance for the following service deployment. Without loss of generality, we define a function $Q(t)$ to measure the queue backlog (ie, queue length) in time slot t, and its value corresponds to the deviation of current cost from the long-term budget constraint. The queue's dynamics evolves in a recursive manner, ie,

$$Q(t+1) = \max\{Q(t) + C(d_t) - B, 0\}. \tag{21}$$

$Q(\cdot)$'s initial value is 0 when $t = 0$, ie, $Q(0) = 0$. In addition, we define a function as the Lyapunov function shown below:

$$\Psi(Q(t)) \triangleq \frac{1}{2}Q^2(t). \tag{22}$$

Function $\Psi(Q(t))$'s value is used to reflect the stability of queue Q, and a lower value here indicates an even more stable queue. We should try to lower the value to keep queue stable and further preserve the budget constraint. We also define a function of $Q(t)$ for the purpose of capturing the drift between arbitrary two consecutive time slots,

$$o(Q(t)) \triangleq \mathrm{E}\left[\Psi(Q(t+1)) - \Psi(Q(t))|Q(t)\right]. \tag{23}$$

Consequently, the objective of long-term service deployment under Lyapunov optimization framework turns into minimizing the supremum bound on the following inequality in every time slot,

$$o(Q(t)) + \gamma \mathrm{E}[T(d_t)|Q(t)] \leq \Omega + Q(t)\mathrm{E}[C(d_t) - B|Q(t)] + \gamma \mathrm{E}\left[T(d_t)|Q(t)\right], \tag{24}$$

where $\Omega = \frac{1}{2}(\sum_{x=1}^{n} C_x(d_{x,t}) - B)^2$, and γ is a weight to achieve the trade-off between latency minimization and cost minimization on each server. As a result, the original service deployment problem is equivalently modeled as the multi-slot minimization problem as follows.

Definition 2 (Multi-slot minimization)**.** Given the same input as Definition 2, the objective is to find d_t in each time slot t to minimize both latency and cost, ie,

$$\begin{aligned} &\arg\min_{d_t} \gamma T(d_t) + Q(t)C(d_t), t \in [1, \Gamma] \\ &s.t. \sum_{i=1}^{m} d_{x,i,t} \leq |S_x|, \forall x \in [1, 2, \dots, n]. \end{aligned} \tag{25}$$

Based on the above definition, our original problem is converted to multiple per-slot minimization sub-problems. Note that each sub-problem can be separately solved. The additional term $Q(t)C(d_t)$ in Equation (25) is set for the purpose of controlling the cost deficit during each per-slot service deployment. Whenever $Q(t)$'s value increases, the cost deficit has to be declined with high priority. Therefore, for the newly defined multi-slot minimization problem, we are still able to enforce the budget constrained latency minimization in each time slot even without knowing the information in all time slots.

4.2 | Online algorithm

In this subsection, we propose an online algorithm called MUSM to solve the multi-slot minimization problem defined above. Since we have divided the long-term optimization problem into multiple per-slot minimization sub-problems, MUSM is consequently designed to iteratively run at every time slot to solve the sub-problem formalized by Equation (25). We assume it runs on the cloud and receives task information from all MEC servers in the current time slot as input. The solution returned by MSUM is a service deployment decision matrix consisting of a set of n deployment decisions $d_{x,t}$ on every server, while the specific decision $d_{x,i,t}$ for each service is just a binary variable. Therefore, the sub-problem is a nonlinear 0-1 programming, and we present an efficient algorithm by leveraging a branch-and-bound technique.[31] The time slot starts from 0, indicating that our system has already initialized the service deployment on every server before task requests arrive. By considering the lack of task information, we notice that the initial service deployment is generated in a random manner as long as it satisfies each server's constraint, ie, $\sum_{i=1}^{m} d_{x,i,t} \leq |S_x|$. The pseudo code of our algorithm is depicted by Algorithm 1.

Algorithm 1 MUSM

```
Input: the set of servers $\{1, 2, \dots, n\}$, the set of services $\{1, 2, \dots, m\}$, the set of task requests $\{R_{x,t} | x = 1, 2, \dots, n, t = 1, 2, \dots, \Gamma\}$, the set of the
    server's constraints $\{S_x | x = 1, 2, \dots, n\}$, budget $B$, channel bandwidth $w$, background noise $\varpi$;
Output: a series of service deployment matrices $\{d_t | t = 1, 2, \dots, \Gamma\}$;
 1: $Q(0) \leftarrow 0$;
 2: for $t \leftarrow [0, \Gamma]$ do
 3:     if $t = 0$ then
 4:         Generate $d_0$ randomly, which satisfies each server's constraint, ie $\sum_{i=1}^{m} d_{x,i,t} \leq |S_x|$;
 5:     else
 6:         Collect the tasks information from every server;
 7:         Solve a nonlinear 0-1 programming formalized by Equation (25), and obtain $d_t$;
 8:     end if
 9:     Calculate $Q(t+1)$ based on Equation (21);
10: end for
11: return the set of $d_t$;
```

5 | JOINT OPTIMIZATION OF SERVICE DEPLOYMENT AND TASK SCHEDULING

We have discussed service deployment in an MEC system and presented an optimization method in the previous sections. In this section, we extend our work by taking into account the effect of task scheduling upon the target performance, and further investigate the joint optimization by introducing task scheduling decision into our framework.

5.1 | Joint optimization analysis

As discussed in the previous sections, we follow a general principle for task scheduling, under which task requests are shared out to another server if and only if the first server does not have the required service. It is simple to work, yet not optimal. For example, service i is assumed to be installed on servers x and y, and both servers may receive a different number of task requests, which leads to different workloads on two servers. If one of servers, say x, affords heavy workload of service i, so its response latency would be high, while on the contrary, server y's workload might be light. Task scheduling depends solely on the type of service that cannot guarantee the best performance.

In order to take into account task scheduling, we define a basic scheduling decision for task r as follows:

$$I_{x,r,t}^{y} \triangleq \begin{cases} 1, \text{task r is scheduled to server y}, \\ 0, \text{otherwise}. \end{cases} \tag{26}$$

$I_{x,r,t}^{y}$ is also a binary variable and decides whether task r is scheduled from server x to server y. We construct a scheduling matrix for all requests first arrived on server x in time slot t (ie, requests on server x sent from end users).

$$I_{x,t} = \begin{bmatrix} I_{x,1,t}^{x} & I_{x,2,t}^{x} & \cdots & I_{x,r,t}^{x} & \cdots \\ I_{x,1,t}^{y} & I_{x,2,t}^{y} & \cdots & I_{x,r,t}^{y} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ I_{x,1,t}^{z} & I_{x,2,t}^{z} & \cdots & I_{x,r,t}^{z} & \cdots \end{bmatrix}_{n \times |R_{x,t}|},$$

where $I_{x,r,t}^{x}$ refers to whether task r is handled locally on server x. Then, we have the set of tasks handled locally:

$$R_{x,t}^{local} = \left\{ r \middle| I_{x,r,t}^{x} = 1 \wedge r \in R_{x,t} \right\}. \tag{27}$$

Figure 4 shows an illustration of tasks shared in and out by applying a specific scheduling decision. Different from our analysis in Section 3.3, where task requests are randomly scheduled among qualified servers, after we introduce scheduling decision, each task request

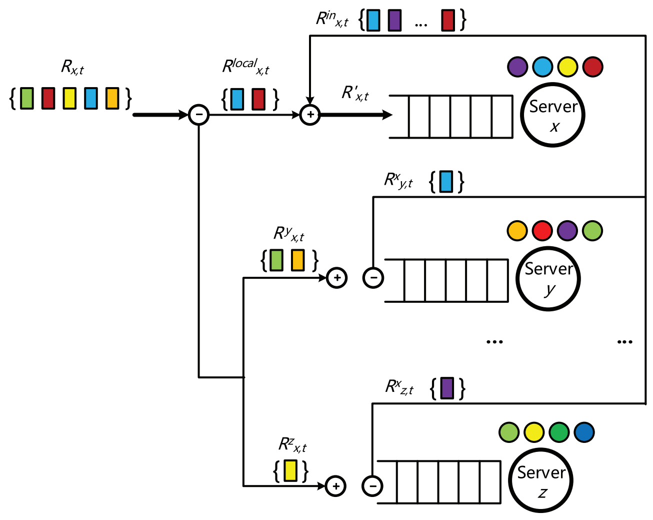

FIGURE 4 Tasks shared in and out by applying a specific scheduling decision

corresponds to a specific decision. For an arbitrary server *y*, the set of tasks scheduled from server *y* to server *x* is represented by

$$R^{x}_{y,t} = \left\{ r \middle| l^{x}_{y,r,t} = 1 \wedge r \in R_{y,t} \right\}. \tag{28}$$

The set of tasks scheduled from other servers to server *x* is

$$R^{in}_{x,t} = \bigcup_{y \neq x} R^{x}_{y,t}. \tag{29}$$

Therefore, the arrival rate of tasks handled locally on server *x* and arrival rate of tasks scheduled to server *x* in time slot *t* are separately represented by

$$\lambda^{local}_{x,t} = \frac{\left|R^{local}_{x,t}\right|}{t}, \lambda^{in}_{x,t} = \frac{\left|R^{in}_{x,t}\right|}{t}. \tag{30}$$

To add $\lambda^{local}_{x,t}$ and $\lambda^{in}_{x,t}$ together, we obtain the total arrival rate of tasks handled on server *x* in time slot *t*, ie, $\lambda'_{x,t}$. Similarly, we can calculate the task handling time on server *x*, ie, $T^{t}_{x-handle}(d_{x,t}, l_{x,t})$, by substituting $\lambda x, t'$ into Equation (3). The average transmission time of tasks shared out from server *x* to server *y* is calculated by

$$T^{t}_{x-y}(d_{x,t}, l_{x,t}) = \frac{\sum\limits_{r \in R^{y}_{x,t}} \pi_r}{\left|R^{y}_{x,t}\right| w_{xy}}. \tag{31}$$

By substituting $T^{t}_{x-y}(d_{x,t}, l_{x,t})$ into Equation (16), we finally obtain the expected latency in time slot *t*, ie, $T(d_t, l_t)$, given a service deployment decision matrix d_t and a task scheduling decision tensor l_t. As a result, the joint optimization problem is defined as follows.

Definition 3 (Dynamic service deployment and task scheduling). In an MEC system, there are *n* MEC servers and *m* services available for deployment. The time is divided into Γ time slots, $T(d_t, l_t)$ and $C(d_t, l_t)$ denote the response latency and total cost respectively, where d_t, l_t denote the service deployment matrix and task scheduling tensor respectively in time slot *t*, $\forall t \leq \Gamma$. The objective is to find d_t and l_t in every time slot *t* to minimize response latency under a given budget constraint *B*, ie,

$$\begin{aligned} &\arg\min \lim_{\Gamma\to\infty} \frac{1}{\Gamma}\sum_{t=1}^{\Gamma} T(d_t, l_t) \\ &s.t. \\ &(1) \lim_{\Gamma\to\infty} \frac{1}{\Gamma}\sum_{t=1}^{\Gamma} C(d_t, l_t) \leq B, \\ &(2) \sum_{i=1}^{m} d_{x,i,t} \leq |S_x|, \forall x \in [1, 2, \ldots, n]. \end{aligned} \tag{32}$$

5.2 | Algorithm

From Definition 2, we notice that the basic decision elements on service deployment and task scheduling, ie, $d_{x,i,t}$ and $l^{y}_{x,r,t}$, are binary variables; thereby, similar to service deployment problem, the joint optimization is also a nonlinear 0-1 programming problem. However, the joint optimization problem is more complicated than service deployment problem because service deployment and task scheduling are highly coupled and it is impossible to achieve both optimal deployment and optimal scheduling at the same time. In this subsection, we give an efficient algorithm called JMUSM, which solves the complex problem in a divide-and-conquer manner. The basic idea is elaborated as follows: at each time slot, we first call MUSM to solve the service deployment problem temporarily and obtain the service deployment solution d_t without coupling task scheduling optimization. Afterwards, we incorporate d_t as known information into the joint optimization problem, and similarly obtain task scheduling solution l_t. Solution l_t is optimal only restricted to given d_t, and there might be a better solution. To achieve the better solution, we design a switching procedure in our algorithm. We randomly choose a pair of MEC servers, and assume that they switch their own service deployment decisions, and then obtain a new scheduling solution l_t. Let ΔT be the difference before and after switching, if the difference value is negative, the switching will not happen; otherwise, it happens with probability $\frac{\Delta T}{T}$, where T denotes the response latency without switching. Here, we choose to conduct deployment update in a probabilistic manner rather than a deterministic manner because the exploitation has a problem of being trapped in a local optimum, and we need to combine exploitation with exploration by letting the proposed algorithm explore a new decision with a certain probability. Specifically, to realize the probabilistic manner, we use a random number generator to output a random variable between 0 and 1, and make comparison between the generated variable with $\frac{\Delta T}{T}$. If the generated variable is lower, the switching is confirmed to conduct; otherwise, the switching is dropped. Consequently, the procedure can guide switching to achieve better latency performance. When all MEC servers have finished the switching procedure, the algorithm stops. The pseudo code of JMUSM is depicted by Algorithm 2.

Algorithm 2 JMUSM

```
Input: the set of servers $\{1, 2, \dots, n\}$, the set of services $\{1, 2, \dots, m\}$, the set of task requests $\{R_{x,t} | x = 1, 2, \dots, n, t = 1, 2, \dots, \Gamma\}$, the set of the
    server's constraints $\{S_x | x = 1, 2, \dots, n\}$, budget $B$, channel bandwidth $w$, background noise $\varpi$;
Output: a series of solution pairs consisting of service deployment matrices and task scheduling tensors $\{(d_t, l_t) | t = 1, 2, \dots, \Gamma\}$;
 1: for $t \leftarrow [0, \Gamma]$ do
 2:     Run MUSM to obtain initial $d_t$;
 3:     Solve the nonlinear 0-1 programming by substituting $d_t$ into Equation (32), and obtain $l_t$;
 4:     for $k \leq \frac{n}{2}$ do
 5:         Choose randomly a pair of unchosen MEC servers $x$ and $y$;
 6:         Switch $d_{x,t}$ and $d_{y,t}$;
 7:         Solve the nonlinear 0-1 programming based on updated $d_t$, and obtain new $l_t$;
 8:         if $\Delta T < 0$ then
 9:             Drop switching;
10:         else
11:             Confirm switching with probability $\frac{\Delta T}{T}$;
12:         end if
13:         $k \leftarrow k + 1$;
14:     end for
15: end for
16: return the set of solution pairs $(d_t, l_t)$;
```

6 | EXPERIMENTAL EVALUATION

In this section, we conduct extensive experiments to validate the performance of our proposed algorithms. The specific setup and results are elaborated below.

6.1 | Setup

In order to begin our experiment, we need to simulate an MEC system. The system is built inside a two-dimensional coordinate, and covers a 2×2 km^2 square area. There are 100 BSs disseminated in this area, and each of them has a serving radius of 200 m. The BS's location depends on its horizontal and vertical coordinates that are randomly drawn from [0 km, 2 km]. There are 200 types of services available for deployment, and the maximum number of services deployed on each server is set to 20, ie, $|S_x = 20|$. The service's size is set to 50 MB. During each time slot, the set of users' tasks arrived in system follows a Poisson distribution with an expected arrival rate at 5000, and each task has its own position, which is also randomly located in the square area. The total number of service types is 200, and each task requires only one service type. Each server is assumed to have a computation power between [20, 30]. For an arbitrary task, its computation demand is uniformly distributed between [1, 50], and its data size ranges from 10 MB to 50 MB. The other parameters are listed in Table 2.

TABLE 2 Default parameter settings

Parameter	Γ	k	B	γ	w	ϖ	$\alpha_{i,t}$	β_t	φ_t
Value	100	2	100	40	5 MHz	−100dBm	0.01	0.005	0.0001

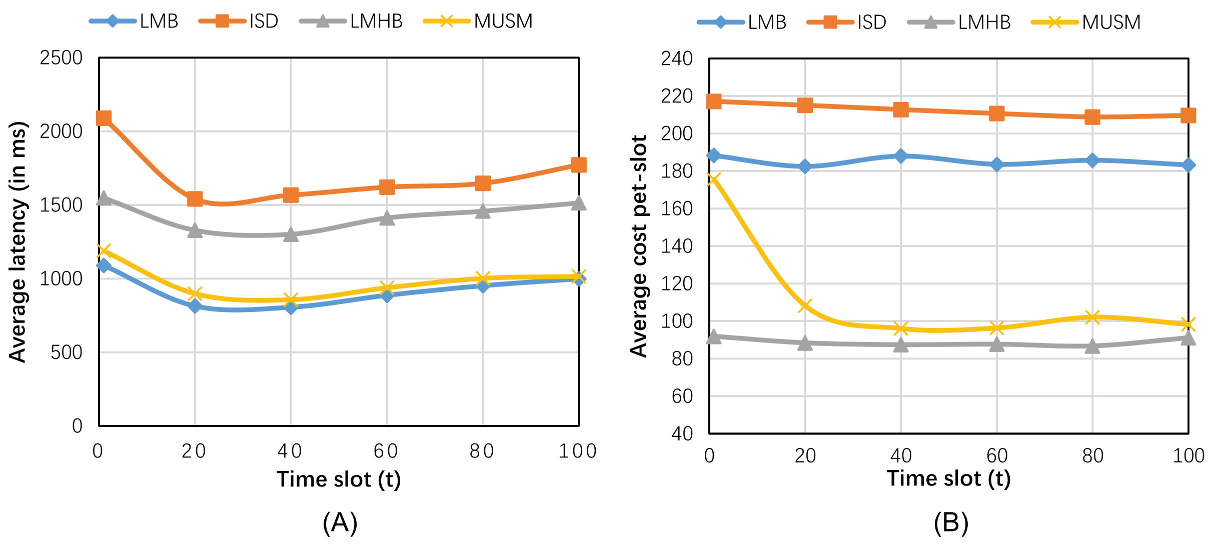

FIGURE 5 Performance comparison among different algorithms

6.2 | Results

6.2.1 | Service deployment comparison

In this group of experiments, we compare our service deployment algorithm MUSM with several benchmarks:

- ***Latency Minimization Without Budget Constraint (LMB)***. This algorithm is designed to solve a variant of our proposed service deployment problem without taking into account the long-term budget constraint.
- ***Independent Service Deployment Without Budget Constraint (ISD)***. This algorithm can be conducted on each server independently, which updates the deployment decision by themselves according to the least recently used (LRU) rule.
- ***Latency Minimization With Hard Budget Constraint (LMHB)***. This algorithm strictly solves the proposed service deployment problem by enforcing a hard budget constraint for per-slot latency minimization.

We conduct all four algorithms respectively within 100 slots of time, and for the sake of fairness, the input of these algorithms in each time slot including the set of tasks arrived on every server, each server's constraint, etc, are set to be identical. The results we observed are shown in Figure 5, where MUSM and LMB implement a similar level of average latency, and they always outperform the other two algorithms. To be specific, LMB performs best in terms of response latency, while its cost is very high due to no budget constraint, and its cost is only slightly lower than that of ISD. In contrast, our algorithm MUSM's cost reduces rapidly to the budget with the increase of time slot, ie, the per-slot cost drops to 100 after running 30 slots. Figure 5A shows that the average latency has a slightly increase along with the drop in cost, ie, 17.8%, but is still better than the performance at the beginning. Among these algorithms, LMHB yields the lowest cost, but its latency is relatively high. It is because that the tasks' computation demands vary differently across time slots; the hard budget constraint enforced by LMHB is not flexible enough to exploit the potential latency reduction by relaxing the constraint temporarily in some slots. ISD uses the LRU method to ensure that each server retains the most recently used services and removed the least used ones, but it neglects the differences of tasks' computation demands and the task sharing among servers. Whenever too many large tasks executed on the server, the latency will definitely increase.

We further make a comparison by adjusting the maximum number of services deployed on a server, ie, $|S_x|$. As illustrated in Figure 6, the latency decreases obviously as more services are allowed to be installed on each server, while cost increases more or less for all algorithms, where MUSM and LMHB show a slight increase. When the maximum is not large enough, few of tasks sent from end users are handled locally on the first arrived servers, which well explains why MUSM and LMB obtain the similar performance on latency. In addition, we observe that MUSM always follows tightly the budget value no matter how the maximum number $|S_x|$ changes. In contrast, the other algorithms do not implement a good trade-off between latency and cost as MUSM achieved.

6.2.2 | Trade-off between latency and cost

Latency and cost are the most important system metrics for the both edge service provider and the end users. Apparently, we cannot minimize both metrics at the same time, and there exists a trade-off between two matrics. In our defined problem, latency is set to be the objective while cost is given as a constraint. Specifically, we introduce a weight γ to implement the trade-off. Figure 7A shows how weight γ affects

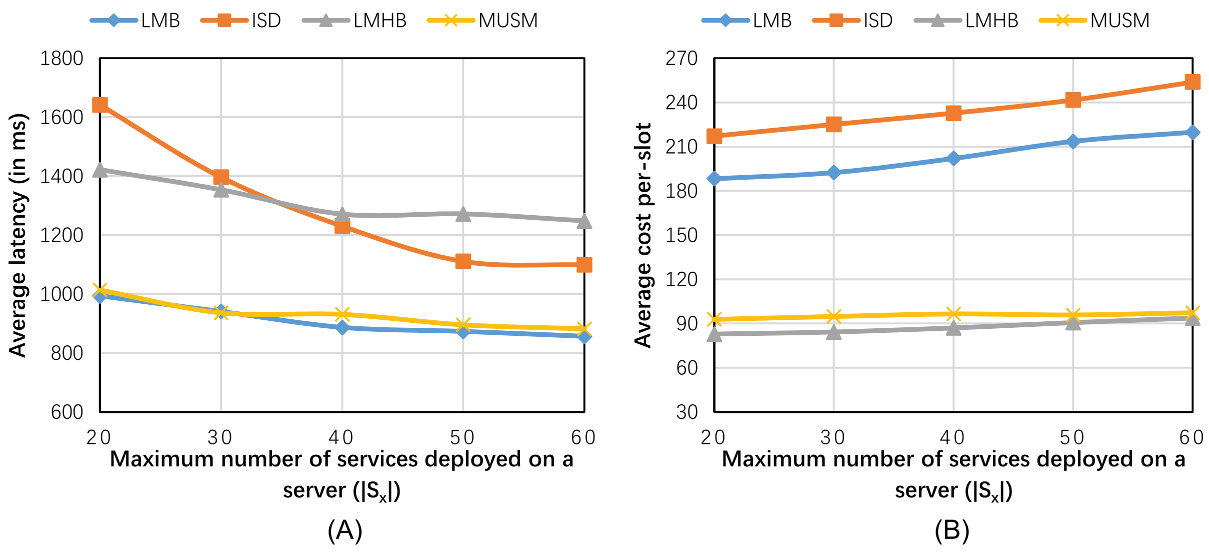

FIGURE 6 Performance comparison under different maximum numbers

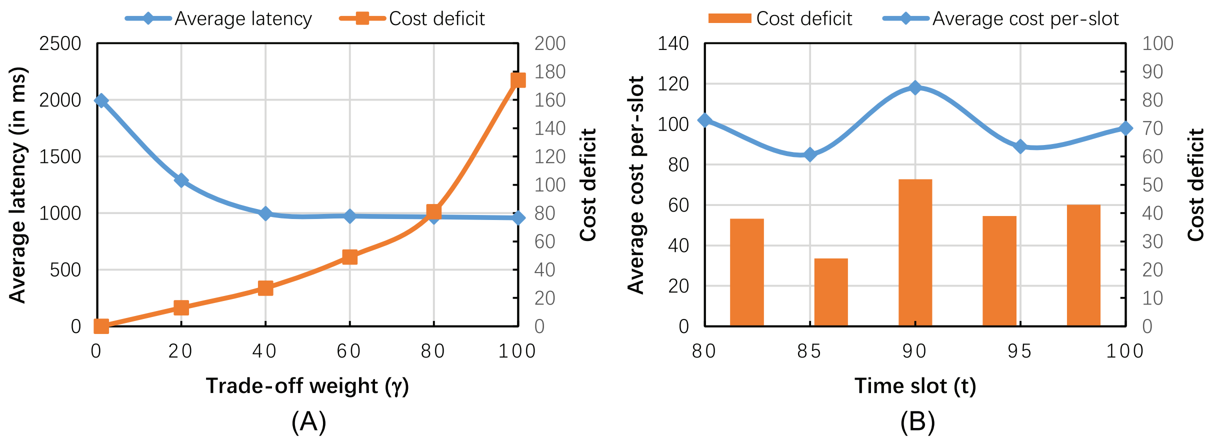

FIGURE 7 The trade-off performance

two system metrics, and we observe a contrary trend on two metrics along with the increase of γ. Based on the Equation (25), we notice that a higher γ indicates the latency minimization has a higher priority to be enforced in solving multi-slot minimization problem. The decreasing trend of latency verifies the conclusion. Meanwhile, the cost deficit increases quickly with γ's growth. Figure 7B further reveals the relationship between cost deficit and average cost per-slot, and the results are reported from slot 80 to slot 100. Whenever cost increases, the cost deficit increases accordingly, and then the cost must decreases so as to satisfy the long-term cost budget. We conclude that just because of our elastic mechanism, the proposed problem can be fully optimized. We can find the suitable point for trade-off by adjusting the budget and weight γ. The adjustment should be conducted based on the actual demand of service providers. To be specific, if the provider has a target level of response latency, it can deduce the expected cost expenditure paid in each time slot, and then, obtain a reference for budget setting. If the budget is fixed, we also could adjust γ's value to approximate the target latency as much as possible. Overall, the primary principle for trade-off adjustment is to achieve even low latency without increasing even more cost.

6.2.3 | The effect of joint optimization

In order to verify the effect of joint optimization, we compare JMUSM with MUSM and MUSM_TS, where MUSM_TS is a simplified implementation of JMUSM without conducting switching procedure. Figure 8 depicts the comparison results in detail. JMUSM and MUSM_TS achieve better system performance than MUSM. It is consistent with our discussion in Section 5.1 that both latency and cost can be further reduced by conducting task scheduling optimization. The difference between JMUSM and MUSM_TS is whether the optimized service deployment is changed or not. It is not difficult to observe from the comparison results that JMUSM works better. Because MUSM_TS conducts task scheduling optimization based on the optimized service deployment, the optimized deployment is generated by MUSM under the assumption of simple task scheduling, so it is not the optimal deployment for the joint optimization. JMUSM exploits the better service deployment by repeatedly conducting service deployment updating based on switching mechanism. Compared with conducting two separate optimizations sequentially, JMUSM joints them together by means of a divide-and-conquer manner, and achieves a great improvement on system performance.

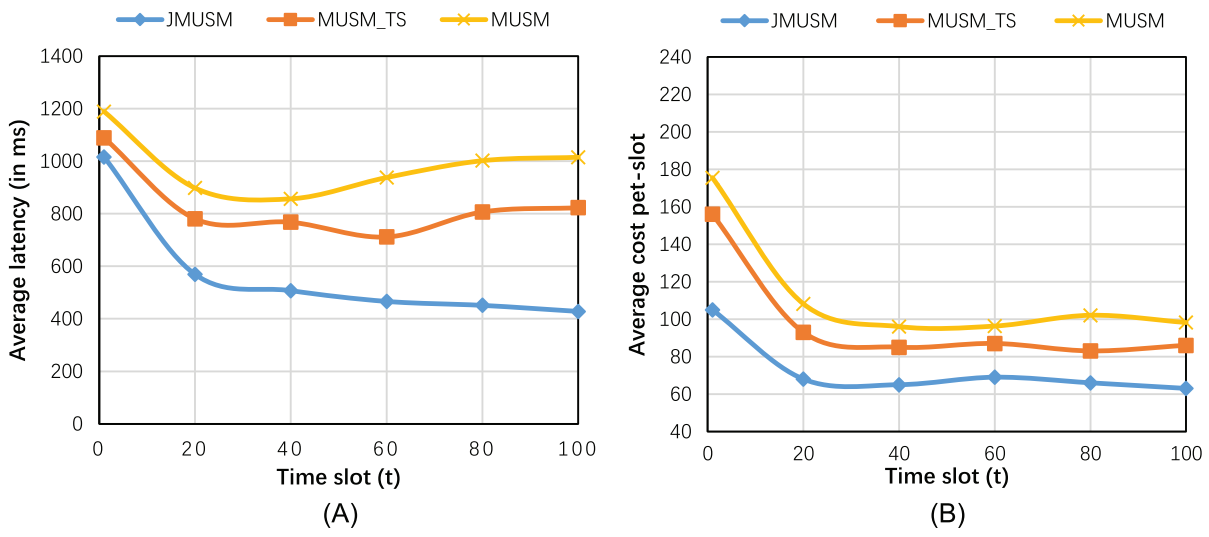

FIGURE 8 Performance comparison on joint optimization

7 | CONCLUSION

In this paper, we investigated the service deployment in a typical MEC system from an edge service provider's perspective. We firstly characterized the major challenges with a comprehensive MEC system model, in which the theoretical analysis of expected latency performance and cost based on giving service deployment decision is presented. Then, we formulated a service deployment problem with the effert to minimize the latency under given budget. To capture the long-term budget constraint, we converted the original problem to multi-slot minimization sub-problems by leveraging Lyapunov framework, and designed an efficient online algorithm accordingly. Furthermore, we combined task scheduling into our framework, and formulated a novel joint optimization problem. Meanwhile a switching-based algorithm is proposed to solve the new problem in a divide-and-conquer manner. Our proposed algorithms are evaluated by extensive experiments.

There are many open issues worth exploring in this field, and we expect our work to foster more future explorations that include but not limited to the following issues.

(1) Service deployment under user-server association, where user and edge server negotiate what proportion of task is offloaded to the server.
(2) Service deployment under one-to-many scenario, where one application task requires more than one service available on the server.
(3) The parallel service deployment algorithm design based on distributed techniques.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (61502328, 61672370, 61572337, 61702351), Jiangsu Planned Projects for Postdoctoral Research Funds (1701173B), the Opening Foundation of Provincial Key Laboratory for Computer Information Processing Technology (KJS1740), and the sponsorship of Jiangsu Overseas Visiting Scholar Program for University Prominent Young & Middle-aged Teachers and Presidents.

ORCID

Jingya Zhou https://orcid.org/0000-0003-0721-7424

REFERENCES

1. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of things (IoT): a vision, architectural elements, and future directions. *Future Generation Comp Syst*. 2013;29(7):1645-1660.
2. Xu J, Ota K, Dong M. Saving energy on the edge: in-memory caching for multi-tier heterogeneous networks. *IEEE Commun Mag*. 2018;56(5):102-107.
3. Wang T, Zhou J, Liu A, Bhuiyan MZA, Wang G, Jia W. Fog-based computing and storage offloading for data synchronization in IoT. *IEEE Internet Things J*. 2018;6:4272-4282.
4. Li H, Ota K, Dong M. Learning IoT in edge: deep learning for the internet of things with edge computing. *IEEE Network*. 2018;32(1):96-101.
5. Satyanarayanan M, Simoens P, Xiao Y, et al. Edge analytics in the internet of things. *IEEE Pervasive Comput*. 2015;14(2):24-31.
6. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. *IEEE Internet Things J*. 2016;3(5):637-646.
7. Zhou Y, Zhang D, Xiong N. Post-cloud computing paradigms: a survey and comparison. *Tsinghua Sci Technol*. 2017;22(6):714-732.
8. Li H, Ota K, Dong M. ECCN: orchestration of edge-centric computing and content-centric networking in the 5G radio access network. *IEEE Wireless Commun*. 2018;25(3):88-93.

9. Barbera MV, Kosta S, Mei A, Stefa J. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In: 2013 Proceedings IEEE INFOCOM; 2013; Turin, Italy.
10. Chen X. Decentralized computation offloading game for mobile cloud computing. *IEEE Trans Parallel Distrib Syst*. 2015;26(4):974-983.
11. Chen X, Jiao L, Li W, Fu X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans Netw*. 2016;24(5):2795-2808.
12. Mao Y, Zhang J, Letaief KB. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J Sel Areas Commun*. 2016;34(12):3590-3605.
13. Pang A-C, Chung W-H, Chiu T-C, Zhang J. Latency-driven cooperative task computing in multi-user fog-radio access networks. Paper presented at: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017; Atlanta, GA.
14. Tan H, Han Z, Li X-Y, Lau FCM. Online job dispatching and scheduling in edge-clouds. Paper presented at: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications; 2017; Atlanta, GA.
15. He T, Khamfroush H, Wang S, Porta TL, Stein S. It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. Paper presented at: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS); 2018; Vienna, Austria.
16. Zhou J, Fan J, Wang J, Jia J. Service deployment for latency sensitive applications in mobile edge computing. Paper presented at: 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD); 2018; Lanzhou, China.
17. Wang L, Jiao L, Li J, Mühlhäuser M. Online resource allocation for arbitrary user mobility in distributed edge clouds. Paper presented at: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017; Atlanta, GA.
18. Jia M, Liang W, Xu Z. QoS-aware task offloading in distributed cloudlets with virtual network function services. In: Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems; 2017; Miami, FL.
19. Chen L, Zhou S, Xu J. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Trans Netw*. 2018;26(4):1619-1632.
20. Shanmugam K, Golrezaei N, Dimakis AG, Molisch AF, Caire G. Femtocaching: wireless content delivery through distributed caching helpers. *IEEE Trans Inf Theory*. 2013;59(12):8402-8413.
21. Prabh KS, Abdelzaher TF. Energy-conserving data cache placement in sensor networks. *ACM Trans Sensor Netw*. 2005;1(2):178-203.
22. Müller S, Atan O, van der Schaar M, Klein A. Context-aware proactive content caching with service differentiation in wireless networks. *IEEE Trans Wireless Commun*. 2017;16(2):1024-1036.
23. Xie Q, Wang Q, Yu N, Huang H, Jia X. Dynamic service caching in mobile edge networks. Paper presented: 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS); 2018; Chengdu, China.
24. Xu J, Chen L, Zhou P. Joint service caching and task offloading for mobile edge computing in dense networks. Paper presented at: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications; 2018; Honolulu, HI.
25. Wang L, Jiao L, He T, Li J, Muhlhauser M. Service entity placement for social virtual reality applications in edge computing. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications; 2018; Honolulu, HI.
26. Dong F, Luo J, Liu B. A performance fluctuation-aware stochastic scheduling mechanism for workflow applications in cloud environment. *IEICE Trans Inf Syst*. 2014;97-D(10):2641-2651.
27. Ge X, Tu S, Mao G, Wang C-X, Han T. 5G ultra-dense cellular networks. *IEEE Wireless Commun*. 2016;23(1):72-79.
28. Rappaport TS. *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ: Prentice Hall; 1996.
29. Chan WC, Lu T-C, Chen R-J. Pollaczek-Khinchin formula for the M/G/1 queue in discrete time with vacations. *IEE Proc-Comput Digit Tech*. 1997;144(4):222-226.
30. Liu Z, Chen Y, Bash C, et al. Renewable and cooling aware workload management for sustainable data centers. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems; 2012; London, UK.
31. Hansen P. Methods of nonlinear 0-1 programming. *Ann Discrete Math*. 1979;5:53-70.

How to cite this article: Zhou J, Fan J, Wang J, Jia J. Dynamic service deployment for budget-constrained mobile edge computing. *Concurrency Computat Pract Exper*. 2019;31:e5436. https://doi.org/10.1002/cpe.5436