Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach

Shangguang Wang, *Senior Member, IEEE*, Yan Guo, Ning Zhang, *Member, IEEE*, Peng Yang, Ao Zhou, and Xuemin Shen, *Fellow, IEEE*

Abstract—As an emerging service architecture, microservice enables decomposition of a monolithic web service into a set of independent lightweight services which can be executed independently. With mobile edge computing, microservices can be further deployed in edge clouds dynamically, launched quickly, and migrated across edge clouds easily, providing better services for users in proximity. However, the user mobility can result in frequent switch of nearby edge clouds, which increases the service delay when users move away from their serving edge clouds. To address this issue, this article investigates microservice coordination among edge clouds to enable seamless and real-time responses to service requests from mobile users. The objective of this work is to devise the optimal microservice coordination scheme which can reduce the overall service delay with low costs. To this end, we first propose a dynamic programming-based offline microservice coordination algorithm, that can achieve the globally optimal performance. However, the offline algorithm heavily relies on the availability of the prior information such as computation request arrivals, time-varying channel conditions and edge cloud's computation capabilities required, which is hard to be obtained. Therefore, we reformulate the microservice coordination problem using Markov decision process framework and then propose a reinforcement learning-based online microservice coordination algorithm to learn the optimal strategy. Theoretical analysis proves that the offline algorithm can find the optimal solution while the online algorithm can achieve near-optimal performance. Furthermore, based on two real-world datasets, i.e., the Telecom's base station dataset and Taxi Track dataset from Shanghai, experiments are conducted. The experimental results demonstrate that the proposed online algorithm outperforms existing algorithms in terms of service delay and migration costs, and the achieved performance is close to the optimal performance obtained by the offline algorithm.

Index Terms—Microservice, mobile edge computing, coordination, delay, migration

1 Introduction

Driven by container technology, microservice architecture can decompose a monolithic web service into a set of lightweight services which can be executed independently [1], [2], [3]. Compared with monolithic architecture, microservice architecture can bring flexibility and scalability because individual microservice can be deployed, migrated, and removed on demand during runtime [4], [5]. Therefore, it has drawn great attention from both academia and industry. For instance, many enterprises are replacing monolithic architecture by microservice architecture, such as Facebook, AT&T, Amazon,[1] Netflix,[2] and Uber.[3] On the network side, mobile edge computing emerges as a new network architecture, where edge clouds are deployed at base stations by endowing the latter with cloud functionalities [6], [7], [8], [9], [10], [11]. Combining microservice architecture with mobile edge computing has attracted much attentions. Microservices in edge clouds can provide proximity services for nearby mobile users with low delay. Moreover, they can be deployed dynamically, launched quickly, and migrated easily on demand.

Along with the aforementioned benefits come the great challenges. Due to the limited coverage of an edge cloud and high user mobility (e.g., autonomous vehicles), the service delay can be greatly deteriorated when users move away from their serving edge clouds [12], [13]. It is of significance yet very challenging to ensure that moving users can still receive services with low delay. However, the majority of existing studies on microservice deployment and scheduling only deal with a static network and fixed service requests [14], [15], [16]. The dynamically changing resource availability and service requests related to user mobility are seldom considered in mobile edge computing environment.

1. https://thenewstack.io/led-amazon-microservices-architecture/

- *S. Wang, Y. Guo, and A. Zhou are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: {sgwang, guoyan, aozhou}@bupt.edu.cn.*
- *N. Zhang is with the Department of Computing Sciences, Texas A&M University-Corpus Christi6300 Ocean Dr., Corpus Christi, TX 78412 USA. E-mail: ning.zhang@tamucc.edu.*
- *P. Yang and X. Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. E-mail: {p38yang, sshen}@uwaterloo.ca.*

Manuscript received 3 Nov. 2018; revised 18 Sept. 2019; accepted 1 Dec. 2019. Date of publication 5 Dec. 2019; date of current version 3 Feb. 2021. (Corresponding author: Ning Zhang.) Digital Object Identifier no. 10.1109/TMC.2019.2957804

2. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

3. https://eng.uber.com/soa/

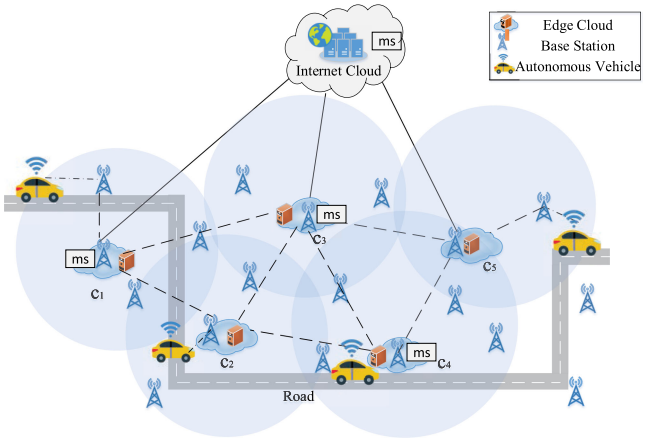


Fig. 1. Illustration of microservice coordination in mobile edge computing.

To address this issue, service migration is proposed, which deals with whether, how, and where the service should be migrated when users change their locations [17], [18], [19], [20]. However, they only focus on migration of a single service, without considering other deployed services that can serve mobile users and fail to fully utilize the dynamic network resources.

In mobile edge computing, the coordination of microservices deployed across edge clouds plays a significant role in providing seamless and real-time services to mobile users. It requires that the running microservices on some edge clouds can be deployed quickly in any edge cloud. The critical question is how to choose the optimal edge cloud to run the microservices for mobile users. Specifically, microservice coordination in mobile edge computing is mainly to determine which edge cloud should perform the microservices, taking into account the user mobility and the dynamics in the network.

A sample scenario is illustrated in Fig. 1, where an autonomous vehicle detects cars, pedestrians, cyclists, road signs, and other objects in real time to make the appropriate control decisions for driving safety [21]. In this scenario, there are one Internet cloud, five edge clouds, nineteen base stations, and one automatic vehicle. An edge cloud is responsible for all base stations located within its coverage, and a base station only connects with a single edge cloud. The object detection microservices are deployed on some edge clouds. The microservices are performed by edge clouds for autonomous vehicle with limited computational capabilities [22], [23]. Suppose that the vehicle has a trajectory shown in Fig. 1. When the vehicle is within the coverage of edge cloud c_1, the object detection can be performed by the microservices in c_1. When the vehicle reaches edge cloud c_2, the object detection can be performed by the original microservices in c_1 with a long communication delay, or by c_3 or c_4 with computation data migration, or by c_2 with quick microservice deployment and computation data migration. Note that, the computation data migration is to migrate the unfinished computation task and intermediate computation state (file system and memory state) to synchronize instances. A microservice should be deployed rapidly if there is no such microservice in the selected edge cloud. Similarly, when the vehicle reaches c_4, the object detection can be performed by c_4 directly, or by other edge clouds with possible computation migration and microservice deployment or by Internet cloud. When the vehicle reaches c_5, the object detection can be performed by any edge cloud or Internet cloud. Therefore, microservice coordination is to coordinate the microservices within different edge clouds to perform real-time object detection during the entire journey of mobile users.

When an autonomous vehicle moves from the coverage of one edge cloud to the coverage of another edge cloud, multiple edge clouds with different computation capabilities are available to either perform the microservices or deploy the microservices. The object detection can be performed in three different ways, i.e., by the previously serving edge cloud, by Internet cloud or the edge clouds deployed with the microservices, or by the current edge cloud with no microservices where the microservices will be deployed. In the first case, there is no migration delay and cost for computation data migration and microservice deployment, but at the expense of communication delay. In the second case, no microservice deployment is needed, but at the expense of communication and migration delay and migration cost. In the third case, the communication delay is lowest, with the expense of potential migration delay and cost. Therefore, there exists a trade-off between the overall service delay and migration cost in microservice coordination. An optimal microservice coordination scheme is urgently needed to coordinate edge clouds for performing the microservice sequentially, in order to reduce the overall delay and migration cost. Note that, microservice coordination for the duration of a mobile user's journey is not a simple repetitive selection of edge clouds. The current coordination decisions may change the microservice deployment, which in turn affect the subsequent decisions.

In this paper, we investigate the microservice coordination problem with the objective of minimizing the overall delay and migration cost. The main contributions of this work are summarized as follows:

1) We first study the microservice coordination with the availability of information regarding the service requests, computational capabilities of the edge clouds, and the channel conditions. We formulate it as a shortest-path problem with a look-ahead time window and propose a dynamic programming-based offline microservice coordination algorithm. Through theoretical analysis, it is proved that such offline algorithm can achieve the optimal solution.
2) Furthermore, we investigate the microservice coordination problem without prior information. We formulate it as Markov decision process and propose a reinforcement learning (RL)-based online microservice coordination algorithm to the optimal decision. Through theoretical analysis, it is proved that the proposed online algorithm can achieve near optimal solution.
3) Experiments are implemented based on two real-world datasets, i.e., Shanghai Telecom's base station dataset and Shanghai Taxi Track dataset. The results demonstrate that the offline microservice coordination algorithm can find the optimal solution with the lowest delay and migration cost while the online

microservice coordination algorithm can achieve close-to-optimal performance.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the system model and problem formulation. Section 4 proposes the dynamic programming-based offline microservice coordination algorithm. Section 5 presents the RL-based online microservice coordination algorithm and the performance analysis. Section 6 provides experimental results. Finally, conclusions are provided in Section 7.

2 Related Work

Microservice architecture has significant benefits in terms of flexibility and scalability, compared with the traditional monolithic architecture. Microservice has been drawing great attentions in the literature [14].

To provide complex services for users using deployed microservices, it is necessary to schedule the distributed microservices in chains appropriately [15], [24]. For example, Bhamare *et al.* [25] studied the scheduling of microservices among multiple clouds, and presented a affinity-based scheduling heuristic method to reduce the overall turnaround time of an end-to-end service and the total traffic generated. Niu *et al.* [16] proposed a microservice chain-oriented load balancing algorithm which balances the working load based on the microservice requirements of chains and minimizes delay. However, these studies on microservice deployment and scheduling mainly focus on static networks and fixed service requests. The presence of dynamically changing resource availability and service requests related to user mobility in a mobile edge computing environment has not been considered sufficiently.

With user mobility, it becomes of great challenges to enable seamless and real-time responses to service requests from mobile users. To address this issue, service migration is proposed, which deals with whether, how, and where the service should be migrated when users change their locations [17], [18], [19], [20]. For example, Yao *et al.* [26] studied the VM migration in Vehicle Ad-Hoc Network, to minimize the overall network cost for migration and normal data traffic. Ksentini *et al.* [27] considered 1-D user mobility model and then formulate the mobility driven service migration problem as a Markov decision process. Then, the mobility model is extended to 2-D mobility in [28], [29]. Taleb *et al.* [29] considered 2-D random walk mobility model and apply a Markov decision process based algorithm for optimizing service migration decisions. Further, Plachy *et al.* [30] exploited prediction of users' movement and proposed an algorithm to find the most suitable data communication path or VM migration. In addition, Zhang *et al.* [31] proposed a deep RL based virtual machine migration approach. Tang *et al.* [32] proposed a container migration algorithm to support mobility tasks with various application requirements. Sun *et al.* [33] developed a user-centric energy-aware service migration scheme by joint optimizing the radio resource and the computational resource.

However, these studies only focus on the migration of a single service, without considering other deployed services that can serve mobile users. They fail to fully utilize network resources. In contrast to the above studies, we focus on microservice coordination in mobile edge computing while considering microservices already deployed in many edge clouds. In our microservice coordination scheme, users not only can access service from the current edge cloud or the edge cloud which the service is migrated to, but also from other edge clouds where the microservices have already been deployed. Different from the migration of a single service, the migration in microservice coordination refers to the migration of unfinished tasks, not specific microservices. The microservices will not disappear with the migration of the tasks of a single user, and can still provide service for other users. The coordination of microservices in different edge clouds aim to find the optimal sequence of edge clouds to perform the microservices for mobile users, which encompasses the selection of edge clouds and dynamic microservice deployment.

3 System Model and Problem Definitions

3.1 System Model

As shown in Fig. 1, we consider a mobile edge network $G = (B \cup C, E)$ with a number of base stations, a set of edge clouds placed at the base stations, and an Internet cloud. Denote B as the set of base stations, C as the set of clouds, and E as the set of links between base stations. In addition, m and n denote the number of base stations and clouds in B and C, respectively. Each base station $b_i \in B(1 \leqslant i \leqslant m)$ is assigned to one and only one edge cloud. All edge clouds together with an Internet cloud can provide service for users. The Internet Cloud host all of the microservices. Each cloud $c_i \in C(1 \leqslant i \leqslant n)$ can be either an edge cloud or an Internet cloud. The microservice configuration in a mobile edge network is denoted by $H = \{h_{i,t}\}(1 \leqslant i \leqslant n, t = 1, ...)$, where $h_{i,t}$ represents whether cloud c_i hosts the microservices at time t. Here, $h_{i,t} = 1$ if cloud c_i hosts the microservices at time t; otherwise, $h_{i,t} = 0$. Note that $\forall t$, $h_{i,t} = 1$ if c_i denotes an Internet cloud. In Fig. 1, there are 19 base stations, five edge clouds, and an Internet cloud. In addition, three microservices have been deployed according to the mobile access pattern in mobile edge computing [34].

In Fig. 1, the autonomous vehicle captures images (video) continuously and then transmits the detection task to nearby edge clouds through the local base stations. Note that we call the edge cloud that is responsible for the local base station of the mobile user the *nearby edge cloud*. For simplicity, we assume that the autonomous vehicle follows a certain path to its destination. Its trajectory is denoted by $L_u = \{l_u(t)\}$, where $l_u(t)$ denotes the location of the vehicle at time t. As the vehicle moves, its local base station switches seamlessly and automatically according to the mobility management protocol communication networks. The switch among base stations leads to the automatic switch of the vehicle's nearby edge cloud. Therefore, we can determine the nearby edge cloud of the user at time t (which is denoted by $e(t)$) according to L_u. To handle the case in which the autonomous vehicle moves from the coverage of one edge cloud to the coverage of another edge cloud, i.e., $e(t) \neq e(t-1)$, a service coordination scheme is needed to determine which edge cloud performs the computation task and whether new microservices should be deployed. The objective of microservice coordination is to minimize the overall delay (consisting

of computation delay, transmission delay, and migration delay) and the migration cost.

3.2 Problem Definitions

3.2.1 Computation Delay

The vehicle captures videos and transmits the entire detection task to edge clouds for computing. The task at time t can be represented by a three-parameter model $A(p_t, \tau_t, w_t)$, where p_t denotes the task input size (in bits), τ_t denotes the completion deadline (in seconds), and w_t is the computational intensity (in CPU cycles per bit). Taking the real-time pedestrian detection application as an example, the vehicle generates a video stream detection task that will be computed on the edge cloud. During each time slot, the vehicle transmits a video clip with a data size of p_t.

Each edge cloud can allocate computational resources for multiple tasks from different users simultaneously by processor sharing. However, the edge clouds have relatively limited computational resources. It is necessary to consider the non-negligible task execution time in a general mobile edge computing system. Let $f_{i,t}$ denote the computational intensity (i.e., CPU frequency) that edge cloud c_i can allocate to the task p_t. If edge cloud c_i is selected to execute the task p_t, the task execution time, denoted by $r_e(i,t)$, can be calculated as

$$r_e(i,t) = \frac{p_t w_t}{f_{i,t}}. \tag{1}$$

Note that the computation delay is defined as infinity if $f_{i,t} = 0$. Furthermore, when the task generated at time t arrives at edge cloud c_i, there may be some unfinished tasks waiting for processing due to the limited computational capacity of edge clouds. Therefore, the queuing time should be taken into account. Then, the total computation delay for task p_t performed by edge cloud c_i is

$$r_c(i,t) = r_e(i,t) + r_q(i,t), \tag{2}$$

where $r_q(i,t)$ denotes the queuing time of task p_t in edge cloud c_i. Let $w^q_{i,t}$ denote the computational intensity of unfinished tasks in edge cloud c_i and $f^q_{i,t}$ denote the computational intensity that edge cloud c_i allocate to the task $w^q_{i,t}$. The queuing time can be computed by $\frac{w^q_{i,t}}{f^q_{i,t}}$.

3.2.2 Communication Delay

The task data transmission can involve three steps: the data transmission from the vehicle to its local base station through the wireless channel; the data transmission from its local base station to the nearby edge cloud; and the data transmission from its nearby edge cloud to the selected edge cloud.

For the wireless channel, let g_t denote the channel gain between the vehicle at location $l_u(t)$ and its local base station $b_t \in B$. Denote S as the transmission power of the vehicle, W as the channel bandwidth, and N as the noise power. Then, the maximum transmission rate is

$$tr_t = W \log_2\left(1 + \frac{Sg_t}{N}\right). \tag{3}$$

In the mobile edge network, let $e_{i,j} \in E$ denote the communication delay of each link between two base stations b_i and b_j. The topological structure of these base stations is

$$E = \begin{Bmatrix} e_{1,1} & \cdots & e_{1,m} \\ \vdots & \ddots & \vdots \\ e_{m,1} & \cdots & e_{m,m} \end{Bmatrix}, \tag{4}$$

where base station b_i and base station b_j are connected if $e_{i,j}$ is finite, otherwise, they are not connected.

Each edge cloud is implemented on a base station equipped with a server based on the existing network topology, so the path between a base station and its edge cloud as well as the path between edge clouds are composed of the links in the weighted graph E. Specifically, the path from local base station b_t to selected edge cloud c_i is the shortest path between b_t and c_i through $e(t)$. Note that if c_i denotes an Internet cloud, the path between b_t and c_i is the path from base station to Internet cloud through the nearby edge cloud $e(t)$ and core network. Let $d(i,t)$ denote the communication delay from the local base station to the selected edge cloud c_i. Then, the overall communication delay of task p_t is given by

$$r_t(i,t) = \frac{p_t}{tr_t} + d(i,t). \tag{5}$$

3.2.3 Migration Delay and Cost

When the vehicle moves and the serving edge cloud switches, an additional delay and cost are incurred because of the computation migration and possible deployment of microservices. Let $x(t) \in C$ denote the edge cloud that serves the tasks at timeslot t. If the microservices cannot be found in $x(t+1)$, i.e., $h_{x(t+1),t} = 0$, the microservices will be deployed rapidly. The unfinished computation task and intermediate data (file system and state data) are then migrated to $x(t+1)$ to synchronize instances.

The process of migrating computation data is inspired by pre-copy memory migration [13][35]. Here, all the migration data are pre transmitted from $x(t)$ to $x(t+1)$ while the microservice is still running until prespecified criteria are met. Then, the running microservice is suspended and the remaining data are transferred to the destination edge cloud. During the computation migration, only the downtime, namely the time interval during which the microservice is not running, affects the user-perceived delay. Therefore, we only consider the downtime of migration, instead of the overall migration time. After the migration, the unfinished task will continue to be processed in edge cloud $x(t+1)$. Since the computation time of unfinished task at the previously serving edge cloud has been counted in Eq. (1), it needs to be recalculated. Therefore, the migration delay consists of the downtime of the migration and the new execution time of unfinished task.

The size of unfinished task at t is

$$p^m_t = min\left\{p_t, max\left\{0, p_t - \frac{\triangle t - r_t(x(t),t) - r_q(x(t),t)}{r_e(x(t),t)} p_t\right\}\right\}, \tag{6}$$

where $\triangle t$ is the length of a time slot. Let $\tilde{p}_t$ denote $p_t - \frac{\triangle t - r_t(x(t),t) - r_q(x(t),t)}{r_e(x(t),t)} p_t$. If the communication delay is longer than a time slot, the $\tilde{p}_t$ is greater than or equal to p_t,

therefore $p_t^m = p_t$ and the whole task p_t should be migrated. If only a part of task can be finished in one time slot, $\tilde{p}_t$ is the size of unfinished task and $p_t^m = \tilde{p}_t$.If the task p_t can be finished in one time slot, i.e., $r_t(x(t),t) + r_c(x(t),t) \leqslant \triangle t$, $\tilde{p}_t$ is less than or equal to 0, therefore $max\{0, \tilde{p}_t\} = 0$ and there is no data need to be migrated.

Therefore, the migration delay of p_t^m is:

$$r_m(x(t), x(t+1), t+1) = \lambda_t \mathbb{I}\{x(t+1) \neq x(t)\} + \frac{p_t^m w_t}{f_{x(t+1),t+1}} - \frac{p_t^m w_t}{f_{x(t),t}}, \tag{7}$$

where λ_t is the downtime of the migration and $\mathbb{I}\{y\}$ is an indictor function. $\mathbb{I}\{y\}$=1 if event y is true and $\mathbb{I}\{y\} = 0$ otherwise. The unfinished task p_t^m is executed in edge cloud $x(t+1)$ rather than the previously serving edge cloud $x(t)$, therefore the execution time of migrated task is $\frac{p_t^m w_t}{f_{x(t+1),t+1}} - \frac{p_t^m w_t}{f_{x(t),t}}$. If the computing power of $x(t+1)$ is more than $x(t)$, $\frac{p_t^m w_t}{f_{x(t+1),t+1}} - \frac{p_t^m w_t}{f_{x(t),t}}$ is less than 0 which means the migration can reduce the execution delay. Otherwise, the migration can increase the execution delay of migrated task. We set $r_m(:,:,1) = 0$ because there is no migration in the first time slot. Note that the effect of both transferred file system and memory state are considered and captured in the downtime. Therefore, we do not need to model them separately.

The cost comprises monetary cost, migration resource consumption, and the cost of the placement of new microservices. The migration cost at time $t+1$ can be expressed as

$$u(x(t), x(t+1), t+1) = \mu(h_{i,t}, x(t), x(t+1)) \mathbb{I}\{x(t+1) \neq x(t)\}, \tag{8}$$

where $\mu(h_{i,t}, x(t), x(t+1))$ is a non-decreasing function of $d(x(t), x(t+1))$. Note that $\mu(:,:,1) = 0$ and $\mu(1,:,:) = 0$.

The microservice configuration of the edge clouds will change according to the microservice deployment, and this change can be expressed as follows:

$$h_{i,t+1} = \begin{cases} 1 & if\ i = x(t+1) \\ h_{i,t} & else. \end{cases} \tag{9}$$

3.2.4 Problem Formulation

The ultimate goal is to find the optimal microservice coordination scheme $x^*(1, \ldots, \infty)$ that minimizes the overall delay and migration cost over a sufficiently long time. The overall delay at time slot t is

$$r(x(t-1), x(t), t) = r_c(x(t), t) + r_t(x(t), t) + r_m(x(t-1), x(t), t). \tag{10}$$

The overall delay of a microservice coordination scheme for vehicle L_u is

$$R(x) = \sum_{t=1}^{\infty} r(x(t-1), x(t), t). \tag{11}$$

The overall migration cost is

$$U(x) = \sum_{t=1}^{\infty} u(x(t-1), x(t), t). \tag{12}$$

Formally, the microservice coordination problem is formulated as follows:

$$\mathbf{P1:} \quad \min_{x(1,\cdots,\infty)} R(x), U(x) \tag{13}$$

$$s.t. \quad r_c(x(t),t) + r_t(x(t),t) \leq \tau_t, \quad \forall t \qquad x(t) \in C, \quad \forall t. \tag{14}$$

However, finding the optimal solution to *P1* is difficult, because it requires the complete information about the entire trip of the vehicle and edge clouds, including the trajectory of the vehicle, the parameters of all tasks, computational intensity of all edge clouds, and traffic intensity of all base stations. Therefore, depending on whether we have the complete information, we devise two coordination algorithms. The first is an offline coordination algorithm where we know the complete future information for a look-ahead time window. The second is an online coordination algorithm in which we only know the current information. In what follows, we will present the offline and online coordination algorithms, respectively.

4 Offline Microservice Coordination

In this section, we focus on the microservice coordination problem when the complete information for the next T slots is available. We devise an offline algorithm to find the optimal solution, which is denoted as x_{off}. The proposed offline algorithm can provide a theoretical upper bound on the performance of any practical online algorithms. Moreover, this offline algorithm can also be applied to the microservice coordination problem with predicted information.

4.1 Offline Problem Formulation

When the look-ahead window size T is given, the overall delay and migration cost of a microservice coordination scheme for each window (containing time slots $t_0, \ldots, t_0 + T - 1$) are as follows:

$$R_T(x) = \sum_{t=t_0}^{t_0-1+T} r(x(t-1), x(t), t), \tag{15}$$

$$U_T(x) = \sum_{t=t_0}^{t_0-1+T} u(x(t-1), x(t), t). \tag{16}$$

The offline microservice coordination problem with a look-ahead window can be stated as follows:

$$\mathbf{P2:} \quad \min_{x_{off}(t_0,\ldots,t_0+T-1)} R_T(x), U_T(x) \tag{17}$$

$$s.t. \quad r_c(x(t),t) + r_t(x(t),t) \leq \tau_t, \quad \forall t \in [t_0, t_0+T-1] \qquad x(t) \in C, \quad \forall t \in [t_0, t_0+T-1]. \tag{18}$$

To obtain an optimal solution, we employ the simple additive weighting model to transform the problem into a single-objective optimization problem with weakly Pareto optimal solutions. The utility function is as follows:

$$\mathbb{F}(x) = \omega \frac{R_T(x) - R_{min}}{R_{max} - R_{min}} + (1-\omega) \frac{U_T(x) - U_{min}}{U_{max} - U_{min}}, \tag{19}$$

where R_{max} and U_{max} represent the maximum value of delay and migration cost during the time window T, respectively. Moreover R_{min} and U_{min} represent the minimum value of the delay and migration cost over time window T, respectively. Weight ω represents the user preference for delay, and $0 \leqslant \omega \leqslant 1$. The optimization problem can then be stated as follows:

$$\mathbf{P3:} \qquad min\ \mathbb{F}(x). \tag{20}$$

The solution of *P3* is a weakly Pareto optimal solution of **P2**. Note that the solution of **P3** can reach Pareto optimal if $\omega \in (0,1)$. The according definitions and proof are as follows.

Definition 1. *Solution $x^* \in C$ is said to be a weakly Pareto optimal solution of **P2**, if and only if there does not exist another $x \in C$ such that $R(x) < R(x^*), U(x) < U(x^*)$.*

Definition 2. *Solution $\hat{x} \in C$ is said to be a Pareto optimal solution of **P2**, if and only if there does not exist another $x \in C$ such that $R(x) \leqslant R_T(\hat{x}), U_T(x) \leqslant U(\hat{x})$.*

Theorem 1. *The solution of* **P3** *is a weakly Pareto optimal solution of* **P2**, *and the solution of* **P3** *is Pareto optimal if the weight coefficient ω is a positive number less than 1.*

Proof. Let x^* be a solution of **P3**. Suppose that it is not a weakly Pareto optimal solution of **P2**. In this case, there exists a solution $x \in C$ such that $R(x) < R(x^*)$ and $U(x) < U(x^*)$. According to the assumption of the weighting coefficients, $0 \leqslant \omega \leqslant 1$. Thus, we have $\omega \frac{R_T(x)-R_{min}}{R_{max}-} R_{min} + (1-\omega)\frac{U_T(x)-}{} U_{min} U_{max} - U_{min} < \omega \frac{R_T(x^*)-R_{min}}{R_{max}-R_{min}} + (1-\omega)\frac{U_T(x^*)-U_{min}}{U_{max}-U_{min}}$. This contradicts the assumption that x^* is a solution of **P3**. That is, x^* is a weakly Pareto optimal solution of **P2**.

If $\omega \in (0,1)$, suppose that x^* is not a Pareto optimal solution of **P2**. This means that there exists a solution $x \in C$ such that $R(x) \leqslant R(x^*)$, $U(x) \leqslant U(x^*)$, and either $R(x) < R(x^*)$ or $U(x) < U(x^*)$. Because $0 < \omega < 1$, we have $\omega \frac{R_T(x)-}{} R_{min} R_{max} - R_{min} + (1-\omega)\frac{U_T(x)-U_{min}}{U_{max}-U_{min}} < \omega \frac{R_T(x^*)-}{} R_{min} R_{max} - R_{min} + (1-\omega)\frac{U_T(x^*)-U_{min}}{U_{max}-U_{min}}$. This contradicts the assumption that x^* is a solution of **P3**. That is, x^* is a Pareto optimal solution of **P2** if $\omega \in (0,1)$. □

4.2 Equivalence to the Shortest-Path Problem

The offline microservice coordination problem is equivalent to a shortest-path problem as shown in Fig. 2. In this study, we consider that the re-selection of edge clouds to perform the microservices and potential migration only occurs when the nearby edge cloud switches. The layers of the shortest-path problem correspond to the switch times of the nearby edge cloud, rather than the actual physical time slots. Each node represents a possible edge cloud, each edge represents a possible selection of edge clouds when the vehicle moves out of the coverage area of the nearby edge cloud, and the weight of each edge is the sum of the utility value from the current slot to the next possible reselection, where the utility value of $x(t)$ is

$$f(x(t),t) = \omega \frac{r(x(t),t) - R_{min}}{R_{max} - R_{min}} + (1-\omega)\frac{u(x(t),t) - U_{min}}{U_{max} - U_{min}}. \tag{21}$$

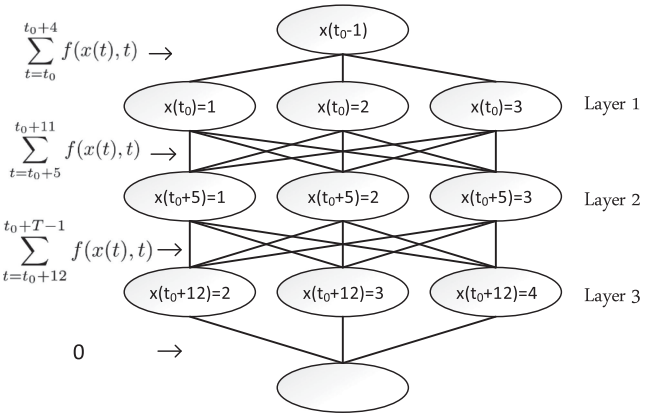


Fig. 2. Offline scenario formulation with $T = 20$, and the nearby edge cloud $e(t)$ switches at slots t_0, $t_0 + 5$, and $t_0 + 12$.

Note that, when we compute the weight of $x(t)$ at time slot t, $x(t-1)$ has been determined, so the utility value f is only related to $(x(t), t)$.

In Fig. 2, there are three candidate edge clouds for $x(t)$, and the switch times of nearby edge cloud is three. The edge cloud at time slot $t_0 - 1$ is given, and the weight of edge $(x(t_0-1), x(t_0))$ is the sum of the utility values from t_0 to $t_0 + 4$. The endpoint is virtual and the weight of each edge to the endpoint is 0. The weight of the shortest path is the minimum utility value $\sum_{t=t_0}^{t_0+T-1} f(x(t),t)$ of all possible microservice coordination schemes, and the shortest path corresponds to the optimal coordination scheme.

4.3 Offline Microservice Coordination Algorithm

Inspired by the approach of dynamic programming, we propose an offline microservice coordination algorithm, called offline algorithm, to solve this shortest-path problem. The offline algorithm is shown in Algorithm 1, where j represents the candidate edge clouds in the previous selection and i represents the candidate edge clouds in the current selection. The values of $F'(j)$ for all j represent the total utility value from time slot t_0 to the current nearby edge cloud switch, $F(i,j)$ for all i,j represents the total utility value from the current switch of nearby edge cloud to the next switch of nearby edge cloud, and $f(i,j,t)$ represents the utility value at time slot t when the previous edge cloud is i and the current edge cloud is j, i.e., $x(t-1) = i, x(t) = j$.

In Algorithm 1, Lines 3–26 iteratively find the shortest path (optimal microservice coordination scheme) for each look-ahead time window T. The iteration starts from the second layer of the graph in Fig. 2. When the nearby edge cloud switches (Line 5), Line 8 selects the optimal edge cloud with respect to the previous selection by solving the Bellman's equation of the problem [36]. For example, in Fig. 2, when $t = t_0 + 12$, F' is the set of the values of the nodes in the second layer, and F is the set of the weights of the edges between the second layer and the third layer. Line 8 selects the optimal edge cloud from the second layer for each node in the third layer. Cloud $x_i(k)$ in Line 9 denotes the optimal edge cloud at the $(k-1)^{th}$ switch of nearby edge cloud for edge cloud i at the kth switch of nearby edge cloud. After iterating from t_0 to $t_0 + T - 1$, the algorithm can determine the optimal microservice coordination scheme for time window T.

Algorithm 1. Offline Algorithm

Input: L_u, E, H; $A(p_t, \tau_t, w_t)$, the computational intensity of all edge clouds and the traffic intensity of all base stations
Output: Optimal microservice coordination scheme $\{x(t)\}$

1: Initialize $t_0 = 1$
2: **loop**
3: Initialize $j = x(t_0 - 1), F(i, j) = f(i, j, t_0), F'(j) = 0, k = 0$
4: **for** $t = t_0 + 1, \cdots, t_0 + T - 1$ **do**
5: **if** $e(t) \neq e(t-1)$ **then**
6: $k = k + 1$
7: **for** all i **do**
8: $j^* = \arg\min_j \{F'(j) + F(i, j)\}$
9: $x_i(k) = j^*$
10: $F'(j) = F'(j^*) + F(i, j^*)$
11: $F(i, j) = f(i, j, t)$
12: **end for**
13: **else**
14: **for** all i, j **do**
15: $F(i, j) = F(i, j) + f(i, j, t)$
16: **end for**
17: **end if**
18: **end for**
19: $k = k + 1$
20: **for** all i **do**
21: $j^* = \arg\min_j \{F'(j) + f(i, j, t)\}$
22: $x_i(k) = j^*$
23: $F'(j) = F'(j) + F(i, j^*)$
24: **end for**
25: $j^* = \arg\min_j \{F'(j)\}$
26: Apply coordination scheme x_{j^*} in slot $t_0, \cdots, t_0 + T - 1$
27: $t_0 = t_0 + T$
28: **end loop**

Consider a more general setting in which the set of candidate edge clouds is time-varying. For example, the edge clouds can be turned on/off according to a sleeping strategy for energy saving [37]. Determining the optimal microservice coordination scheme for a varying edge cloud set is of great challenge. However, in Algorithm 1, the set of alternative edge clouds for each $x(t)$ can be different. Therefore, this algorithm can also be applied to a varying edge cloud set.

4.4 Performance Analysis

In this subsection, we show that the result of the offline algorithm is the optimal solution of **P3** and analyze its time complexity.

Theorem 2. *For the offline microservice coordination problem with a look-ahead time window, Algorithm 1 provides the optimal solution.*

Proof. Define $p_{k,n}$ as the decision sequence of edge cloud $x(t)$ during the sub-process from the kth layer to the end, and define $p^*_{k,n}$ to be the result of the offline algorithm for this sub-process. Define $\mathbb{V}_{k,n}$ as the overall utility function of $p_{k,n}$. According to $j^* = \arg\min_j \{F'(j) + F(i, j)\}$ and $F'(j) = \arg\min_j \{F'(j) + F(i, j)\}$ in Lines 8 and 10, we have

$$\mathbb{V}_{1,k+1}(p^*_{1,k+1}) = min_{p_{k,k+1}} \{\mathbb{V}_{1,k}(p^*_{1,k}) + \mathbb{V}_{k,k+1}(p_{k,k+1})\}. \tag{22}$$

It means that the shortest path from the beginning to the $(k+1)$th layer is the minimum sum of the shortest path from the beginning to the kth layer and the path from the kth layer to the $(k+1)$th layer.

For the utility function of the overall process $\mathbb{V}_{1,n}$ of all possible schemes $p_{1,n}$, according to Eq. (22), we have

$$\begin{aligned}
\mathbb{V}_{1,n}(p_{1,n}) &= \mathbb{V}_{1,n-1}(p_{1,n-1}) + \mathbb{V}_{n-1,n}(p_{n-1,n}) \\
&\geq min_{p_{1,n-1}} \{\mathbb{V}_{1,n-1}(p_{1,n-1})\} + \mathbb{V}_{n-1,n}(p_{n-1,n}) \\
&= \mathbb{V}_{1,n-1}(p^*_{1,n-1}) + \mathbb{V}_{n-1,n}(p_{n-1,n}) \\
&\geq min_{p_{n-1,n}} \{min_{p_{1,n-1}} \{\mathbb{V}_{1,n-1}(p_{1,n-1})\} \\
&\quad + \mathbb{V}_{n-1,n}(p_{n-1,n})\} \\
&= \mathbb{V}_{1,n}(p^*_{1,n}).
\end{aligned} \tag{23}$$

Therefore, the result of the offline algorithm $p^*_{1,n}$ is the optimal solution of the offline microservice coordination problem with a look-ahead time window. □

The time-complexity of Algorithm 1 is $O(|S|^2 T)$, where $|S|$ is the number of states, because there are at most $|S|^2$ candidate state-action pairs in Lines 7–8 and 15, and there can be at most $|S|^2 * T$ possible selection schemes.

5 Online Microservice Coordination

In this section, we consider the scenario in which only the current information is known and we propose an online algorithm to solve **P1** without future information by learning the future reward of current decisions.

5.1 Equivalence to Markov Decision Process

The microservice coordination problem is a sequential decision problem and can be formulated within a Markov decision process framework.

The state at time slot t is denoted by $s(t) = \{x(t), h(t)\}$, where $x(t)$ is the edge cloud that performs the computation tasks and $h(t)$ is the microservice instance configuration of all edge clouds. Policy π, which makes control decisions at any state $s(t)$, is a probability distribution on the action set, and we use $a(s(t)) \in C$ to represent the action of selecting a new edge cloud for $s(t)$. This action causes the system to transmit to a new state $s(t+1) = s'(t) = (x(t+1), h(t+1)) = a(s(t))$. The immediate reward of action $a(s(t))$ is $R^a_s = -f(s(t), a, t)$, where $f(s(t), a, t)$ is the utility value at time $t+1$ when current state is $s(t)$ and the selected edge cloud for next time slot is a. Starting from initial state $s(0) = s_0$, the long-term cumulative reward function is

$$G_\pi(0) = \sum_{k=0}^{\infty} \gamma^k R^{a_\pi}_{s(k)}, \tag{24}$$

where $\gamma \in [0, 1]$ is the discount factor.

The objective of the microservice coordination problem is to find the policy π with the maximum expected cumulative reward, and we denote the expected value of cumulative reward G_π at state s by state-value function

$$V_\pi(s) = E\left[\sum_{k=0}^{\infty} \gamma^k R^{a_\pi}_{s(k)} | s(0) = s\right]. \tag{25}$$

Moreover, the optimal state-value function is

$$V^*(s_0) = \max_{\pi} V_\pi(s_0). \tag{26}$$

Theorem 3. *The optimal solution is the stationary policy obtained by Bellman's optimal equation* $V^*(s) = \max_a\{R_s^a + \gamma \sum_{s'} P_{s,s'}^a V^*(s')\}$.

Proof. From the definition of the state-value function, we know that

$$\begin{aligned} V(s) &= E[G(t)|s(t)=s] \\ &= E\Big[R_{s(t)}^a + \gamma(R_{s(t+1)}^a + \gamma R_{s(t+2)}^a + \cdots)|s(t)=s\Big] \\ &= E\Big[R_{s(t)}^a + \gamma G(t+1)|s(t)=s\Big] \\ &= E\Big[R_{s(t)}^{a_\pi} + \gamma V(s(t+1))|s(t)=s\Big]. \end{aligned} \tag{27}$$

The last equation is true because

$$\begin{aligned} V(s(t)) &= E_{s_t,s_{t+1},\cdots}(R_{s_t}^a + \gamma G(t+1)) \\ &= E_{s_t}(R_{s_t}^a + \gamma E_{s_{t+1},\cdots}(G(t+1))) \\ &= E_{s_t}(R_{s_t}^a + \gamma V(s(t+1))). \end{aligned} \tag{28}$$

Therefore, $V_\pi(s) = \sum_a \pi(a|s)(R_s^a + \gamma \sum_{s'} P_{s,s'}^a V_\pi(s'))$, where $\pi(a|s)$ is the probability of taking action a at state s, and $P_{s,s'}^a$ is the probability of transition from s to s' by a. According to the definition of an optimal state-value function, $V^*(s) = \max_a\{R_s^a + \gamma \sum_{s'} P_{s,s'}^a V^*(s')\}$. □

5.2 RL-Based Online Microservice Coordination Algorithm

To find the optimal policy of the microservice coordination problem based on Markov decision process framework, we propose an RL-based online microservice coordination algorithm, called online algorithm, as shown in Algorithm 2. RL is concerned with how an agent ought to take actions in an environment so as to maximize the cumulative reward. The agent in microservice coordination can learn the long-term cumulative reward of current selections by trial-and-error interaction with the dynamic mobile edge environment.

The online algorithm is inspired by the Q-learning algorithm, which is one of the most popular RL methods that is applied in many research areas, such as service migration [34] and computation offloading [38]. In Algorithm 2, Lines 5–10 form an iteration in which the agent takes action at current state s to move to the next state s' by receiving an immediate reward $\hat{R}$ for updating the Q-table. Here, $Q(s,a)$ in the Q-table represents the expected long-term reward of taking action a at state s. Line 6 selects the actions for state s according to the ϵ-greedy policy. The specific ϵ-greedy policy used here is

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & if\ a = arg\max_a Q(s,a) \\ \frac{\epsilon}{|A(s)|} & else \end{cases}, \tag{29}$$

where $|A(s)|$ is the number of alternative actions for state s. The ϵ-greedy policy balances exploitation and exploration strategies, where exploitation selects the action with the maximum Q value and exploration selects the other actions. Reward $\hat{R}$ in Line 7 is the predicted reward provided by the future reward parameter prediction module according to the historical data of edge clouds and base stations. The specific prediction method used in this study records the parameters and regards the mean of the records up to the current time slot as parameters for the future reward. The iteration is terminated when $Q(s,a)$ converges. The online algorithm considers not only the immediate reward of current decisions, but also the future possible rewards to determine the optimal microservice scheme along the way.

Algorithm 2. Online Algorithm

Input: $l_u(t)$, $e(t)$, $h(t-1)$, $A(p_t, \tau_t, w_t)$, the computational intensity of all edge clouds, and traffic intensity of all base stations at t
Output: Optimal microservice coordination scheme

```
1: if e(t) ≠ e(t−1) then
2:    s_0 = (x(t−1), h(t−1)), Q = 0
3:    repeat
4:       s = s_0
5:       for i = t, ··· , T do
6:          Select a based on ε-greedy policy
7:          Observe s′, predict R̂
8:          Q(s,a) = Q(s,a) + α[R̂ + γ max_a′ Q(s′,a′) − Q(s,a)]
9:          s = s′
10:      end for
11:   until Q(s,a) converge
12:   x(t) = arg max_a Q(x(t−1), a)
13: else
14:   x(t) = x(t−1)
15: end if
```

5.3 Performance Analysis

In this section, we analyze the performance of the online algorithm for **P1** in terms of the convergence and sample complexity, which indicate how much experience the algorithm needs to learn for a given task [39].

The essence of Algorithm 2 is to approximate V using experience learned from environment by updating the value of Q and then make decisions according to the approximate value function. First, we prove the convergence of the online algorithm.

Theorem 4. *The approximate state-value function in Algorithm 2 converges to a value that is at least locally optimal.*

Proof. We represent the approximate value function Q as a parameterized linear functional form with weight vector θ and the corresponding feature vector $\boldsymbol{\phi}$: $Q = \theta^T \boldsymbol{\phi}_s$. It is natural to use semi-gradient method updates with a linear function approximation, the gradient of the approximate value function with respect to θ is $\nabla Q = \boldsymbol{\phi}_s$. The update of the approximate value function reduces to the update of the linear weight vector as follows:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha(R_{t+1} + \gamma\theta_t^T\boldsymbol{\phi}_{t+1} - \theta_t^T\boldsymbol{\phi}_t)\boldsymbol{\phi}_t \\ &= \theta_t + \alpha(R_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^T\theta_t), \end{aligned} \tag{30}$$

where we use the notational shorthand $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t)$. Once the approximate value function reaches the steady state,

the expected next weight vector for any θ_t can be written as follows:

$$\begin{aligned} E[\theta_{t+1}|\theta_t] &= \theta_t + \alpha(\mathbf{b} - \mathbf{A}\theta_t) \\ &= (\mathbf{I} - \alpha\mathbf{A})\theta_t + \alpha\mathbf{b}, \end{aligned} \tag{31}$$

where $\mathbf{b} = E[R_{t+1}\phi_t]$ and $\mathbf{A} = E[\phi_t(\phi_t - \gamma\phi_{t+1})^T]$.

When $\mathbf{A}$ is positive definite, then α is set to a value that is smaller than one over the largest value of the diagonal elements of $\mathbf{A}$, $(\mathbf{I} - \alpha\mathbf{A})$ is a matrix whose elements are between 0 and 1. Then, $(\mathbf{I} - \alpha\mathbf{A})\theta_t$ tends to the shrink θ_t, and stability is assured. Therefore, the linear approximation can converge. In the continuing case with $\gamma < 1$, matrix $\mathbf{A}$ can be written as

$$\begin{aligned} \mathbf{A} &= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{s'} p^a_{s,s'} \phi(s)(\phi(s) - \gamma\phi(s'))^T \\ &= \sum_s \mu(s) \sum_{s'} p(s'|s)\phi(s)(\phi(s) - \gamma\phi(s'))^T \\ &= \sum_s \mu(s)\phi(s)(\phi(s) - \gamma \sum_{s'} p(s'|s)\phi(s'))^T \\ &= \phi^T \mathbf{D}(\mathbf{I} - \gamma\mathbf{P})\phi, \end{aligned} \tag{32}$$

where $\mu(s)$ is the stationary distribution under π, $\mathbf{P}$ is the $|S \times S|$ probability matrix, and D is the $|S \times S|$ diagonal matrix with $\mu(s)$ on its diagonal.

From the above formulation, the inner matrix $\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})$ is key to determining the positive definiteness of $\mathbf{A}$. Next, we focus on the positive definiteness of the key matrix.

The row vector of the column sums of $\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})$ can be written as

$$\begin{aligned} \mathbf{1}^T\mathbf{D}(\mathbf{I} - \gamma\mathbf{P}) &= \mu^T(\mathbf{I} - \gamma\mathbf{P}) \\ &= \mu^T - \gamma\mu^T\mathbf{P} \\ &= \mu^T - \gamma\mu^T \\ &= (1 - \gamma)\mu, \end{aligned} \tag{33}$$

where $\mathbf{1}$ is the column vector with all components equal to 1. In addition, $\mu = \mathbf{P}^T\mu$ because μ is the stationary distribution. All components of the key matrix $\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})$ are positive; thus, its column sums are nonnegative. The row sums of the inner matrix are all positive because $\mathbf{P}$ is a stochastic matrix and $\gamma < 1$. It is well known, any matrix M is positive definite if and only if the symmetric matrix $S = M + M^T$ is positive definite [40]. Moreover, any symmetric real matrix is positive definite if all of its diagonal entries are positive and greater than the sum of the corresponding off-diagonal entries [41]. Therefore, key matrix $\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})$ is positive definite and $\mathbf{A}$ is positive definite. Therefore, the approximate value function can converge.□

Finally, we analyze the sample complexity of Algorithm 2, which indicates the number of iterations needed to achieve the ϵ-optimal. It is defined as follows.

Definition 3. *For any fixed $\epsilon > 0$, the sample complexity of the exploration of an RL algorithm is the number of timesteps t such that the policy at time t π_t satisfies $V^{\pi_t}(s_t) < V^*(s_t) - \epsilon$.*

The sample complexity of Algorithm 2 is $O(|S||A|/(\epsilon^4(1-\gamma)^8))$, where $|S|$ and $|A|$ are the number of states and number of actions, respectively [39]. This reveals that the required number of iterations is proportional to the discount factor γ and is inversely proportional to the optimal gap ϵ.

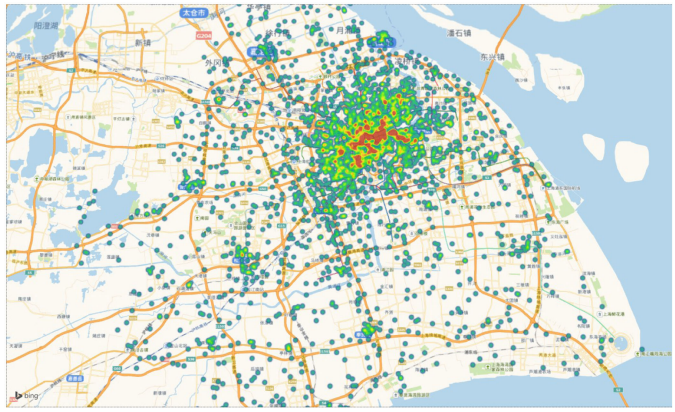


Fig. 3. Distribution of all 3,233 base stations in Shanghai. Each node denotes a base station.

6 Experiment and Results

In this section, we compare the proposed Offline Algorithm and Online Algorithm with several existing coordination algorithms in terms of the overall delay and migration cost.

6.1 Dataset Description

To evaluate the performance of the proposed algorithms, two real-world datasets are used in experiments: Shanghai Telecom's base station dataset and Shanghai Taxi Track dataset.

Shanghai Telecom's base station dataset contains the exact location information of 3,233 base stations and the Internet access information of mobile users that passed through these base stations. More specifically, the dataset contains more than 7.2 million records of Internet accesses through 3,233 base stations from 9,481 mobile phones during six successive months. Each record contains the detailed start time and end time of the base station access for each mobile user. Fig. 3 shows the distribution of the 3,233 base stations in Shanghai, where each node denotes a base station.

The Shanghai Taxi Track dataset contains the tracks of all 4,328 taxis in Shanghai on February 20, 2007. Each track of a taxi contains its specific location information, recorded every about one or half minutes over the whole day. In addition, this dataset contains the direction and instantaneous speed of the taxi and whether the taxi is transporting passengers. Fig. 4 shows four tracks randomly sampled from the dataset, each distinguished by four different colors.

To measure the performance of our microservice coordination algorithms, we combined the above two datasets to simulate a scenario where mobile users call services continuously as they move. Fig. 5 shows how Shanghai Telecom's base station dataset and the Shanghai Taxi Track dataset were combined. Blue points represent the track of a taxi, green points represent the base stations, red points represent the edge clouds, and areas filled with different colors represent the coverage of the edge cloud. The taxi moves through many base stations and hence moves from the coverage area

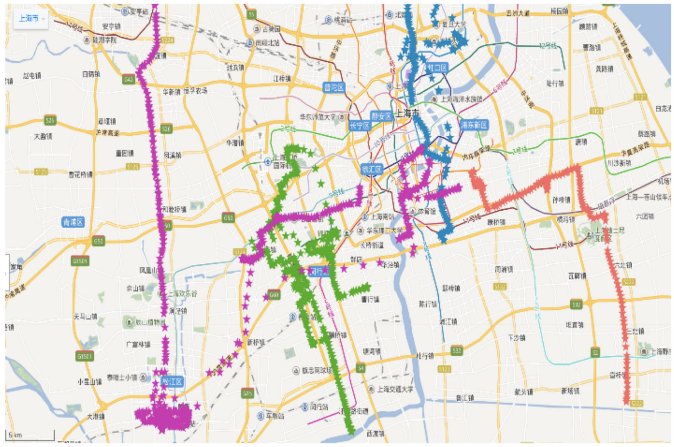


Fig. 4. Tracks of four taxis selected randomly from all taxis in Shanghai. Each color represents a taxi track over a day.

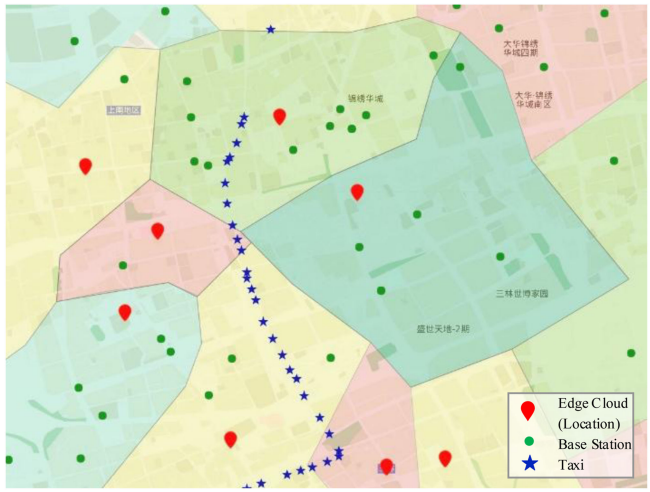


Fig. 5. Detailed track of a taxi.

of one edge cloud to that of another. This figure shows the nearby edge cloud switches as the user moves.

6.2 Experiment Setup

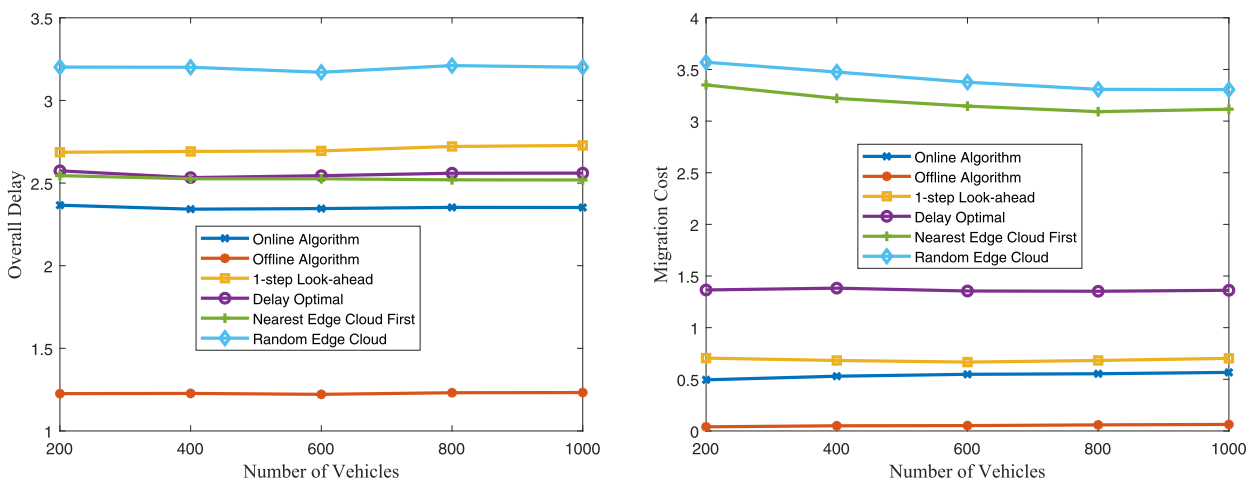
From the 3,233 real base stations in Shanghai, we chose 400 base stations as the placement locations for the edge clouds using a K-means clustering algorithm [42]. We then deployed 150 microservices on these edge clouds randomly.

According to [33], [43], each edge cloud is equipped with multiple CPU cores, and the sum frequency is 25 Ghz. The computation resources that an edge cloud can allocate to a vehicle follows uniform distribution with $f_{i,t} \in [0, 25]$ Ghz. The computation resources that the Internet cloud can allocate to a vehicle is 25 GHz. The wireless channel gain g_t is $127 + 30 \times \log d$, where d is the distance between the vehicle at location $l_u(t)$ and its local base station $b_t \in B$ [44]. The channel bandwidth W is set to 20 Mhz, the noise power N is 2×10^{-13}W, and the transmitted power of vehicle S is 0.5 W. The delay of each link $e_{i,j} \in E$ is randomly generated in the range 5 to 50 ms. The communication delay between any base station and the Internet cloud is 50 ms. The queuing time of tasks in each edge cloud are randomly generated in the range 0 to 5 ms. The downtime of the migration λ_t is 5ms, and the migration cost $\mu = d(x(t), x(t-1)) \times (1 - h_{i,t})$.

The surrounding environment (e.g., the density of base stations, the distribution of edge clouds and the microservice configuration) and the track (e.g., the speed) of each vehicle are different. To improve the reliability of experiment results, we use Monte Carlo method and execute 1000 simulations by optimizing the coordination for a vehicle in 1000 scenes. We randomly selected the tracks of 1000 taxis to serve as 1000 scenes and only used 30 minutes of the track for the experiments. All the resulting data in the experiments are the average values of the vehicles. We consider that the tasks generated during the entire track follow a Poisson distribution with a mean of 0.6 Mbits/s, the computational intensity w_t is uniformly distributed within [500,1000] cycle/bits, and the completion deadline τ_t is 150 ms. All the experiments were conducted on the same computer with an Intel(R) Xeon (R) 2.4 GHz processor with 32.0 GB of RAM, using MATLAB with source code.

We compare the proposed Offline Algorithm and Online Algorithm with four other existing algorithms:

1) *1-step Look-ahead*: In this algorithm, we consider both delay and migration cost, and the vehicle always selects the edge cloud with the minimum utility value



(a) Overall delay (b) Migration cost

Fig. 6. Performance of different algorithms with respect to number of vehicles.

TABLE 1
Distribution of the Switch Times of Vehicles

Switch times	1	2	3	4	5	6	7	8	9	10
Number of vehicles	96	145	166	157	137	100	76	57	27	12

in the current time slot. It is an offline microservice coordination algorithm with a 1-step look-ahead time window.

2) *Delay Optimal*: In this algorithm, we only consider the delay of the microservices, and the vehicle always selects the edge cloud with minimum delay, disregarding the migration cost.
3) *Nearest Edge Cloud First*: In this algorithm, the vehicle always selects the nearby edge cloud and the computation data are migrated along with the switch of nearby edge cloud. The microservices are deployed if there are no microservices in the nearby edge clouds.
4) *Random Edge Cloud*: In this algorithm, the vehicle always selects the next edge cloud randomly.

6.3 Effect of the Number of Vehicles on Delay and Migration Cost

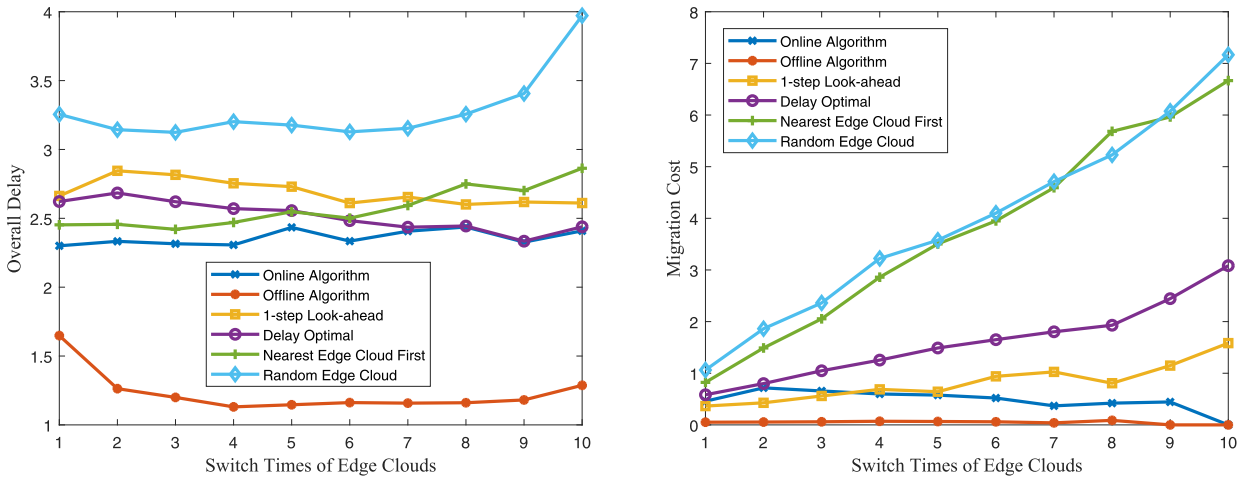
In this subsection, we compare the proposed Offline Algorithm, Online Algorithm, and four comparison algorithms in terms of the delay and migration cost with respect to the number of vehicles from 200 to 1,000.

Figs. 6a and 6b show the performance of delay and migration cost, respectively. The Offline Algorithm achieves the best results for both delay and migration cost due to the utilization of future system information. Among all the algorithms, the results of the Online Algorithm are the closest to that of Offline Algorithm. In fact, its migration cost is just slightly worse than that of the Offline Algorithm. Compared with the 1-step Look-ahead algorithm, the Online Algorithm performs better in terms of delay and migration cost because it considers the impact of future rewards on current decisions. The delay performances of the Delay Optimal algorithm and the Nearest Edge Cloud First algorithm are similar, and they are better than those of the 1-step Look-ahead algorithm because they only focus on the delay rather than both the delay and migration cost. In addition, this causes the migration cost of the Delay Optimal algorithm and Nearest Edge Cloud First algorithm worse than that of the 1-step Look-ahead algorithm. The performances of all the algorithms remain stable as the number of vehicles increases, which indicates that the Offline Algorithm and Online Algorithm can be applied to a large number of vehicles.

6.4 Effect of Switch Times on Delay and Migration Cost

To study the impact of the switch times of the nearby edge clouds on the performance of Offline Algorithm and Online Algorithm, we executed the algorithms while fixing the number of vehicles at 1,000. We then classified the vehicles according to the switch times of nearby edge clouds in their track. The distribution of the switch times of vehicles is shown in Table 1. A higher switch times indicates that the speed of the vehicle is higher or the deployment of the edge clouds in the district is more dense, which increases the likelihood of selecting a new edge cloud.

Fig. 7 shows the performance evaluation results of different algorithms with respect to the increasing switch times of nearby edge clouds. Figs. 7a and 7b illustrate the effects on the performance of delay and migration cost, respectively. We can see that the delay and migration cost of Online Algorithm and Offline Algorithm remain almost the same as the switch times increases. And the migration cost of other algorithms increases with increasing switch times, because the switch between edge clouds leads to the reselection of microservices and increases the likelihood of microservice deployment and computation migration. The overall delay of the Delay Optimal algorithm and 1-step Look-ahead algorithm slightly decreases as the switch times increases. This is because the increasing instances of reselection diminish the impact of future reward. The migration cost of the Nearest Edge Cloud First algorithm increases rapidly because the service instance must be migrated if the nearby edge cloud does not host the microservices. The migration cost of the Nearest Edge Cloud First algorithm also increases because it does not take into account the migration cost.



(a) Overall delay (b) Migration cost

Fig. 7. Performance of different algorithms with respect to the switch times.

In summary, in a smart city, e.g., Shanghai, the Online Algorithm can provide vehicles better experience than the other four representative approaches in terms of delay and migration cost. It is close to the globally optimal performance provided by the Offline Algorithm.

7 Conclusion

In this paper, we have investigated the microservice coordination problem in mobile edge computing environments, to select optimal edge clouds for performing the microservices as a mobile user moves. To reduce the overall delay and migration cost, we have proposed an offline algorithm to find the optimal microservice coordination when future system information is available. Furthermore, we have also proposed an online algorithm that does not require future system information. Through theoretical analysis, it is proved that the offline algorithm can find the optimal solution while the online algorithm can achieve near optimal performance. Moreover, based on two real datasets, the experiments are conducted, which demonstrate that the proposed online algorithm outperforms several representative algorithms in terms of delay and migration cost, and the performance is close to that of the globally optimal solutions provided by the offline algorithm. For the future work, we will jointly investigate microservice coordination and load balancing among microservices in the multiuser mobile edge computing systems.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Grant 61922017 and 61602054) and the BUPT Excellent Ph.D. Students Foundation CX2019211.

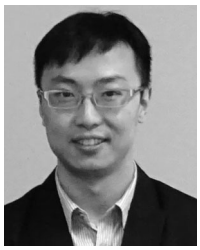
References

[1] A. Sill, "The design and architecture of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 76–80, Sep./Oct. 2016.
[2] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019, doi: 10.1109/TMC.2018.2871842.
[3] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *Proc. IEEE Int. Conf. Softw. Architecture*, 2017, pp. 21–30.
[4] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: Current and future directions," *ACM SIGAPP Appl. Comput. Rev.*, vol. 17, no. 4, pp. 29–45, 2018.
[5] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proc. IEEE 9th Int. Conf. Service-Oriented Comput. Appl.*, 2016, pp. 44–51.
[6] M. Li, R. Yu, P. Si, and Y. Zhang, "Energy-efficient machine-to-machine (M2M) communications in virtualized cellular networks with mobile edge computing (MEC)," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1541–1555, Jul. 2019, doi: 10.1109/TMC.2018.2865312.
[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 4Q 2017.
[8] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.
[9] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Oct.-Dec. 2017.
[10] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 21, no. 4, pp. 915–929, Apr. 2019.
[11] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. S. Shen, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2903240.
[12] Y. Wang, W. Shi, and M. Hu, "Virtual servers co-migration for mobile accesses: Online versus off-line," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2576–2589, Dec. 2015.
[13] K. Ha *et al.*, "Adaptive VM handoff across cloudlets,"CMU School Comput. Sci., Pittsburgh, PA, Tech. Rep. CMU-CS-15–113, 2015.
[14] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May-Jun. 2016.
[15] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 81–88, Sep./Oct. 2016.
[16] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *Proc. 4th IEEE Int. Conf. Comput. Commun.*, 2018, pp. 1–9.
[17] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.
[18] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, no. , pp. 205–228, 2015.
[19] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[20] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.
[21] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2017, pp. 446–454.
[22] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Energy efficient dynamic offloading in mobile edge computing for Internet of Things," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2898657.
[23] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, "Toward efficient content delivery for automated driving services: An edge computing solution," *IEEE Netw.*, vol. 32, no. 1, pp. 80–86, Jan./Feb. 2018.
[24] C. Guerrero, I. Lera, and C. Juiz, "Resource optimization of container orchestration: A case study in multi-cloud microservices-based applications," *J. Supercomputing*, vol. 74, no. 7, pp. 2956–2983, Jul. 2018.
[25] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Multi-objective scheduling of micro-services for optimal service function chains," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–6.
[26] H. Yao, C. Bai, D. Zeng, Q. Liang, and Y. Fan, "Migrate or not? exploring virtual machine migration in roadside cloudlet-based vehicular cloud," *Concurrency Comput.: Pract. Experience*, vol. 27, no. 18, pp. 5780–5792, 2015.
[27] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 1350–1354.
[28] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. 14th IFIP Netw. Conf.*, 2015, pp. 1–9.
[29] T. Taleb, A. Ksentini, and P. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369–382, Apr.-Jun. 2019.
[30] J. Plachy, Z. Becvar, and E. C. Strinati, "Dynamic resource allocation exploiting mobility prediction in mobile edge computing," in *Proc. IEEE 27th Annu. Int. Symp. Personal Indoor Mobile Radio Commun.*, 2016, pp. 1–6.
[31] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 111–118, 2019.
[32] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.
[33] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

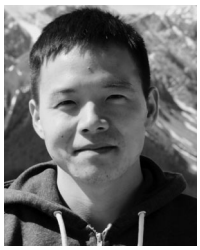
[34] S. Cao, Y. Wang, and C. Xu, "Service migrations in the cloud for mobile accesses: A reinforcement learning approach," in *Proc. 12th Int. Conf. Netw. Architecture Storage*, 2017, pp. 1–10.
[35] H. Assasa, S. V. Yadhav, and L. Westberg, "Service mobility in mobile networks," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, 2015, pp. 397–404.
[36] R. A. Howard, "Dynamic programming," *Manage. Sci.*, vol. 12, no. 5, pp. 317–348, 1966.
[37] L. Chiaraviglio, F. Cuomo, M. Listanti, E. Manzia, and M. Santucci, "Fatigue-aware management of cellular networks infrastructure with sleep modes," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3028–3041, Nov. 2017.
[38] J. Li, Q. Liu, P. Wu, F. Shu, and S. Jin, "Task offloading for UAV-based mobile edge computing via deep reinforcement learning," in *Proc. IEEE/CIC Int. Conf. Commun.*, 2018, pp. 798–802.
[39] A. L. Strehl, L. Li, and M. L. Littman, "Reinforcement learning in finite MDPs: PAC analysis," *J. Mach. Learn. Res.*, vol. 10, pp. 2413–2444, 2009.
[40] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.
[41] R. S. Varga, *Matrix Iterative Analysis*. vol. 27, Berlin, Germany: Springer, 2009.
[42] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.
[43] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[44] C. Niu, Y. Li, R. Q. Hu, and F. Ye, "Fast and efficient radio resource allocation in dynamic ultra-dense heterogeneous networks," *IEEE Access*, vol. 5, pp. 1911–1924, 2017.

Shangguang Wang received the PhD degree from the Beijing University of Posts and Telecommunications, in 2011. He is a professor and vice-director at the State Key Laboratory of Networking and Switching Technology (BUPT). He has published more than 100 papers, and played a key role at many international conferences, such as general chair and PC chair. His research interests include service computing, cloud computing, and mobile edge computing. He is a senior member of the IEEE and the editor-in-chief of the International Journal of Web Science.

Yan Guo received the bachelor degree in mathematics and applied mathematics from the Beijing University of Posts and Telecommunications, in 2016. Currently, she is working toward the PhD degree in computer science at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include Service Computing and Mobile Edge Computing.



Ning Zhang received the PhD degree from the University of Waterloo, Canada, in 2015. He is an assistant professor at Texas A&M University-Corpus Christi, USA. After that, he was a postdoc research fellow at the University of Waterloo and University of Toronto, Canada, respectively. He serves/served as an associate editor of IEEE Access and IET Communication, an area editor of Encyclopedia of Wireless Networks (Springer) and Cambridge Scholars, a guest editor of Wireless Communication and Mobile Computing, International Journal of Distributed Sensor Networks, and Mobile Information System. He also served as the workshop chair for the first IEEE Workshop on Cooperative Edge. He is a recipient of the best paper awards at IEEE Globecom 2014 and IEEE WCSP 2015, respectively. His current research interests include next generation mobile networks, physical layer security, machine learning, and mobile edge computing. He is a member of the IEEE.



Peng Yang received the BE and PhD degrees from the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China, in 2013 and 2018, respectively. He is currently a postdoctoral fellow with the Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada. His research focuses on software defined networking, network function virtualization, and mobile edge computing.

Ao Zhou received the BS, MS, and PhD degrees from the Beijing University of Posts and Telecommunications, Beijing, China, in 2009, 2012 and 2015, respectively. She is currently an associate professor at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published more than 20 research papers. She played a key role at many international conferences, such as PC chair. Her research interests include Cloud Computing and Edge Computing.

Xuemin (Sherman) Shen received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, in 1990. He is currently a University professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is a registered professional engineer of Ontario, Canada, an Engineering Institute of Canada fellow, a Canadian Academy of Engineering fellow, a Royal Society of Canada fellow, and a distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society. He received the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015, and the Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006 and the Outstanding Performance Award in 2004, 2007, 2010, and 2014 from the University of Waterloo and the Premiers Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He served as the technical program committee chair/co-chair for the IEEE Globecom16, IEEE Infocom14, IEEE VTC10 Fall, IEEE Globecom07, the symposia chair for IEEE ICC10, the tutorial chair for the IEEE VTC11 Spring, the chair for the IEEE Communications Society Technical Committee on Wireless Communications, and P2P Communications and Networking. He is the editor-in-chief of the *IEEE Internet of Things Journal* and the vice president on Publications of the IEEE Communications Society. He is a fellow of the IEEE.