Distributed and Dynamic Service Placement in Pervasive Edge Computing Networks

Zhaolong Ning, *Senior Member, IEEE*, Peiran Dong, Xiaojie Wang, Shupeng Wang, Xiping Hu, Song Guo, *Fellow, IEEE*, Tie Qiu, *Senior Member, IEEE*, Bin Hu, *Senior Member, IEEE*, and Ricky Y. K. Kwok, *Fellow, IEEE*

Abstract—The explosive growth of mobile devices promotes the prosperity of novel mobile applications, which can be realized by service offloading with the assistance of edge computing servers. However, due to limited computation and storage capabilities of a single server, long service latency hinders the continuous development of service offloading in mobile networks. By supporting multi-server cooperation, Pervasive Edge Computing (PEC) is promising to enable service migration in highly dynamic mobile networks. With the objective of maximizing the system utility, we formulate the optimization problem by jointly considering the constraints of server storage capability and service execution latency. To enable dynamic service placement, we first utilize Lyapunov optimization method to decompose the long-term optimization problem into a series of instant optimization problems. Then, a sample average approximation-based stochastic algorithm is proposed to approximate the future expected system utility. Afterwards, a distributed Markov approximation algorithm is utilized to determine the service placement configurations. Through theoretical analysis, the time complexity of our proposed algorithm is linear to the number of users, and the backlog queue of PEC servers is stable. Performance evaluations are conducted based on both synthetic and real trace-driven scenarios, with numerical results demonstrating the effectiveness of our proposed algorithm from various aspects.

Index Terms—Pervasive edge computing, service migration, Lyapunov optimization, distributed Markov approximation

1 Introduction

Over the past decades, the explosive growth of mobile devices promoted the prosperity of novel mobile applications. The emerging of augmented reality, online interactive game and real-time video processing attracts and entertains the crowd [1]. In general, these novel applications require powerful computation capabilities and intensive computation resources. Due to the limitation of physical sizes and weights, existing mobile devices cannot be equipped with advanced processors and large volume batteries. Thus, it is difficult to support latency-sensitive applications. The conflict between resource-demanding applications and resource-limitation devices yields an obstacle for the continuing development of mobile applications.

The proliferation of mobile computing can alleviate the burden of portable devices. Specifically, Mobile Cloud Computing (MCC), together with Long Term Evolution (LTE) networks, has received widespread attention [2]. By providing sufficient computation and storage resources for Mobile Users (MUs), MCC enables complicated applications to be executed on mobile devices. However, due to geographical separation between the cloud infrastructure and MUs, they may suffer from long communication latency, deteriorating the Quality of Service (QoS).

Recently, Mobile Edge Computing (MEC) has emerged as a promising paradigm to reduce the communication latency significantly by providing proximal offloading services for MUs. Many existing researches assume that service requests of MUs can be accomplished within a determined time slot [3], [4], [5]. However, in reality, MUs can move out of the wireless coverage of edge servers before service requests are completed. Without the cooperation of nearby edge servers, the backhaul transmission of service results can be interrupted. Thus, services need to be dynamically migrated among multiple edge servers to guarantee service performance. In addition, when confronted with the surge of service-requirements, a single MEC server is hard to meet those requirements with given latency constraints. Dynamic service placement across multiple MEC servers

- *Zhaolong Ning is with the School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China, and also with the School of Software, Dalian University of Technology, Dalian 116024, China. E-mail: z.ning@ieee.org.*
- *Peiran Dong is with the School of Software, Dalian University of Technology, Dalian 116024, China. E-mail: peiran_dong@outlook.com.*
- *Xiaojie Wang is with the School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China, and also with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: xiaojie.kara.wang@ieee.org.*
- *Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: song.guo@polyu.edu.hk.*
- *Shupeng Wang is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100864, China. E-mail: wangshupeng@iie.ac.cn.*
- *Tie Qiu is with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin 300072, China. E-mail: qiutie@ieee.org.*
- *Xiping Hu and Bin Hu are with the School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China. E-mail: {huxp, bh}@lzu.edu.cn.*
- *Ricky Y. K. Kwok is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong, China. E-mail: Ricky.Kwok@hku.hk.*

Manuscript received 11 July 2020; revised 10 Dec. 2020; accepted 13 Dec. 2020. Date of publication 21 Dec. 2020; date of current version 8 Jan. 2021.
(Corresponding authors: Xiaojie Wang and Shupeng Wang.)
Recommended for acceptance by Y. Yang.
Digital Object Identifier no. 10.1109/TPDS.2020.3046000

can relieve the mess by enhancing the robustness of offloading services.

With the objective of providing pervasive offloading services, multiple MEC servers can cooperate to construct Pervasive Edge Computing (PEC) networks [6]. During the whole procedure of offloading (i.e., from raw data uploading to results backhaul downloading), the service needs to be migrated dynamically among PEC servers to follow the erratic movements of MUs. A trade-off arises by considering whether to migrate a service from one server to another: Migrating a service incurs additional transmission overhead, whereas not migrating may lead to backhaul transmission interruption. In addition, since multiple MUs share PEC resources, the service placement configurations of MUs are inter-dependent. For each MU, there is a trade-off between the communication distance and the number of MUs that occupy the same PEC server. Therefore, it should be carefully configured whether to migrate and where to place the service.

A few existing researches have investigated dynamic service placement in mobility scenarios [7], [8], [9], [10]. However, all those researches either require global network information or make service placement policies without considering the long-term effects. In general, it is non-trivial to predict the long-term behavior patterns of MUs, such as movement trajectories. Besides, due to the limitation of communication and computation resources, service placement configurations of MUs are inter-dependent. Therefore, the trade-off between the current service execution cost and future possible migration cost should be carefully coped with.

Based on partial offloading, dynamic service placement enables multi-server cooperation to accomplish one request together. In order to keep the consistency and atomicity of tasks as well as guarantee the robustness of MEC services, all input and output data (also known as a service Virtual Machine (VM)) are migrated from one PEC server to another. This setting has been widely applied in existing researches [11], [12], [13]. Ma *et al.* [14] investigate the handoff mechanism of offloading services, and propose a live migration scheme based on the layered characteristics of the storage system. In this paper, we mainly focus on the service placement scheduling (i.e., when and where to migrate the service VM), and the detailed procedure of request data handoff is not our main consideration. More details can be found in [15].

In this paper, we construct a dynamic service placement framework for PEC networks. By extending preliminary researches, we consider a realistic mobility scenario, where multiple MUs keep moving erratically across the coverage of PEC servers, and contend for computation resources. In particular, under the constraints of service execution delay and limited storage capacity of PEC servers, we propose a Dynamic storAge-Stable Service placement (DASS) algorithm to maximize the system utility, while striking a balance between system stability and overhead. First, we introduce Lyapunov-drift-plus-penalty function to decompose the long-term optimization problem into a series of online problems. Then, to avoid myopic decisions, Sample Average Approximation (SAA) algorithm is utilized to approximate the long-term accumulated system utility. Finally, we formulate the online system utility maximization issue as a Markov approximation optimization problem, and a Markov chain is constructed to calculate the optimal service placement configuration. The main contributions of this paper are summarized as follows:

- We construct a dynamic service placement framework for efficient offloading in PEC networks. A realistic mobility scenario is considered where MUs move erratically across the coverage of PEC servers. Service requests of MUs are generated by following Bernoulli process.
- With the objective of maximizing the long-term system utility while guaranteeing the stability of server storage queues, we utilize Lyapunov optimization to decompose the long-term system utility maximization issue into a series of online Lyapunov drift-plus-penalty minimization problems.
- Without the prior knowledge of future movement trajectories of MUs, the future system utility is approximated by Monte-Carlo based stochastic sampling. Then, we introduce service placement probability distribution and convex log-sum-exp function, to transform the system utility maximization problem into a Markov approximation optimization problem. A distributed Markov chain is constructed to obtain the optimal placement configuration.
- Theoretically analysis and performance evaluation demonstrate the effectiveness and efficiency of our proposed method in terms of system utility, service fraction and convergence time. Besides, the average number of fulfilled requests per time slot is satisfactory, and the storage capability of PEC servers can be fully utilized.

The rest of this paper is organized as follows. Related work is reviewed in Section 2. Section 3 illustrates the system model and formulates the optimization problem. Section 4 proposes a dynamic service placement algorithm, and Section 5 presents theoretical analysis. Performance evaluations are illustrated in Section 6, followed by the conclusion in Section 7.

2 Related Work

In this section, we first present a brief introduction of the state-of-the-art researches on PEC and dynamic service placement. Then, the distinction between existing work and this paper is specified.

2.1 Pervasive Edge Computing

There emerges various applications since the paradigm of PEC has been proposed. One of the key branch is data sharing among edge devices in the PEC environment. In order to improve data availability in mobility scenarios, Huang *et al.* [16] propose an efficient caching algorithm for peer data sharing. Considering load balance among heterogeneous peer edge devices, the authors design fairness metrics to realize fair and low-latency data sharing. With ultra-low transmission delay in the PEC environment, Chen *et al.* [6] develop an online peer data sharing framework to maximize the long-term system revenue, and utilize Lyapunov optimization method to minimize the individual energy consumption. A multi-user computation offloading problem is studied in [17], in which the authors utilize pervasive radio access networks to implement distributed data offloading. A game theoretic approach is adopted to make an energy-efficient offloading decision. Theoretical analysis derives the upper bound of the convergence time and the price of anarchy. The above studies consider relatively static scenarios, where users' locations remain stable during data sharing or offloading procedure.

To adapt changing environment incurred by novel applications and mobility scenarios, it is essential to construct a versatile and robust PEC architecture. Roy *et al.* [18] study high-level context-aware PEC by low-level sensor data collection. Different from single context determination, the proposed energy-aware capturing framework is able to determine multiple context metrics simultaneously. In such cases, the accuracy and efficiency of the context estimation can be improved while the increased overhead of data streaming is tolerable. Li *et al.* [19] design a cost effective network structure to integrate core servers, PEC servers and switches. The novel network provides a power-efficient routing algorithm to improve the availability of data centers. It is suitable for mobility scenarios, since the network topology can be adjusted conveniently. Although the above literatures can be applied in mobility scenarios, they either rely on context-aware architectures or a powerful data center, which leads to extra communication cost.

2.2 Service Migration

Traditional static offloading policies cannot satisfy the requirements of mobility scenarios. Thus, it is necessary to investigate service migration and placement in PEC environments. Wang *et al.* [12] study dynamic service migration under uniform 1-D and 2-D mobility scenarios. The authors utilize Markov decision process to formulate the service migration problem. A policy-iteration algorithm is proposed to find the optimal migration decisions with cubic time complexity. Badri *et al.* [20] jointly consider the QoS and energy consumption in MEC-enabled service placement. The QoS depends on the latency and data sizes of the service requests, and an energy budget is set to avoid the waste of computation resources. The authors propose a modified SAA algorithm to solve the formulated multi-stage stochastic program. He *et al.* [21] formulate the service placement issue as the generalization of the bin packing problem. The available computation capacities of PEC servers are specified by a vector containing four elements, i.e., CPU, memory, storage and bandwidth. To solve the NP-hard problem, the authors propose a potential game-based algorithm to reach the Nash equilibrium. Ouyang *et al.* [22] propose the paradigm of follow-me at the edge, where services are migrated among MEC servers to follow the mobility of users. Since the trajectories of users are unpredictable, Lyapunov optimization is utilized to decompose the long-term cost minimization issue into a series of instant optimization problems. Then, the authors propose both centralized and distributed algorithms to approximate optimal solutions. Wu et al. [11] study service migration based on social characteristics. They construct a multi-factor graph learning model to predict user location. The service virtual machine is migrated to follow user's mobility to provide PEC service. Ma *et al.* [14] investigate the offloading service handoff, and propose a live migration scheme based on the layered characteristics of the storage system. Although many researches have investigated service migration from the aspects of multidimensional maps, QoS and energy-efficiency, few studies consider long-term storage queue stability during service migration.

2.3 Data Replica Placement

Data replica placement is a classical technique for popular content delivery. Replica placement algorithms for content delivery networks can be broadly classified into three categories: QoS aware, consistency aware and energy aware [23]. Yu and Pan [24] jointly consider associations of data-node and node-node for distributed datacenters. A hypergrapgh based problem is formulated to optimize multiple objectives, including routing latency as well as traffic and storage costs. They propose an iteratively hypergraph partition algorithm to obtain the data routing and placement decisions. With the objective of minimizing data-user distance, Aral *et al.* [25] propose a distributed data replication approach based on geographical locations of both data and users. By deciding when and where to create and store a replica, they focus on data dissemination rather than offloading. Most existing researches on replica placement attempt to minimize response latency and maintain high access reliability. Chang *et al.* [26] study the data consistency overheads among multiple replicas, and define two functions to calculate latency and reliability of replica operations. A leader selection algorithm and a replica placement algorithm are developed to minimize leader assignment latency and consistency overheads.

There are some differences between service migration and data replication. First, service migration is for one user rather than common demands. Specifically, data replication usually takes content popularity into account, since it aims to provide service for masses of users. However, service migration is implemented to provide low-latency computing services for MUs, whose requests can be various, such as augmented reality, online interactive game, real-time video processing and remote health monitoring. Second, in the case of stateful services, update operation in data replication results in extra overheads. While in service migration, the same task is not replicated to multiple servers. Only one replication of each task exists on PEC servers at one time. Thus, there is no need for data synchronization. Third, popular data such as videos are replicated to multiple servers to satisfy users' concurrency demands, i.e., multiple users communicate with various servers for the same content simultaneously. However, for each service request, only one determined user demands for results, and communicates with one PEC server at one time.

Referring to the advantages of existing work, we construct a distributed and dynamic service placement framework, where the system utility can be maximized while stabilizing the storage queues of PEC servers. Specifically, the future utility is approximated by utilizing an improved SAA algorithm. In addition, to develop an online algorithm, we avoid complicated iterative operations by a distributed Markov approximation approach. The proposed DASS algorithm is able to achieve a stationary Markov distribution within satisfactory convergence time.

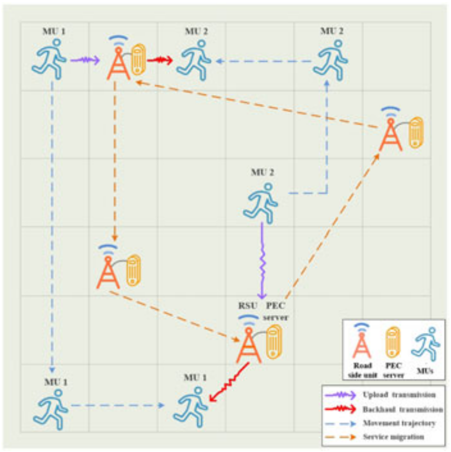


Fig. 1. An illustration of the stochastic mobility system.

3 System Model

As shown in Fig. 1, we consider a stochastic mobility system, where a set of PEC servers, $\mathcal{K} = \{1, 2, \ldots, K\}$, provide computation services for a set of MUs, $\mathcal{N} = \{1, 2, \ldots, N\}$. Each PEC server is attached to a RoadSide Unit (RSU) or a wireless access point via high speed fiber communication. Facilitating to capture the mobility of MUs, the service is assumed to be offered in a discretized time-slotted frames, denoted by $\mathcal{T} = \{0, 1, \ldots, T-1\}$. We distinguish between fast and slow based on the number of required time slots and servers to accomplish one task. If a task can be accomplished by merely one server within one time slot, the scenario can be regarded as a slow scenario, where users move relatively slow, and the offloading process consumes a short time. By contrast, if multiple servers need to cooperate to accomplish a task by service migration during several time slots, the scenario can be regarded as a fast scenario, where users move fast across the coverage area of multiple servers, and the communication for offloading maintains for a relatively long time. In this paper, we main focus on fast scenarios. During the time between two independent tasks, MUs do not access to servers. When MUs generate a new task, the PEC network is constructed by clock synchronization. The information synchronization time can be constrained within an acceptable range by limiting the number of both PEC servers and users. In such cases, the access during each task can be maintained.

MUs communicate with PEC servers through Orthogonal Frequency Division Multiple Access (OFDMA). When multiple MUs access one PEC server at the same time, the communication channel is divided into sub-channels. Each MU occupies one sub-channel, and there is no interference among multiple MUs. In each time slot, the proposed DASS algorithm is performed to determine the service placement. Then, MUs access to the selected PEC servers, respectively. When a MU moves out of the communication coverage of the current accessed server, our proposed algorithm is performed to select the next server to be connected, which is also consistent with the destination of the service migration.

Many existing researches assume that the arrival of service requests follows a determined distribution, so that they can utilize statistical regularity to predict the future states of service arrivals [27], [28], [29]. However, the performance of those algorithms can be significantly reduced once the service arrival distribution in the real-world deviates from the assumed one [30]. Therefore, we consider a stochastic process, where service requests are generated by following Bernoulli process [31]. A service request can be specified by $\{d_i, C_i, T_i^{max} | i \in \mathcal{N}\}$, where d_i denotes the data size, and C_i is the required number of CPU cycles to accomplish the service. Variable T_i^{max} represents the tolerable maximum number of time slots for request processing. Since PEC servers may consume more than one slot (denoted by Δt) to accomplish each request, we assume the uploading and backhaul transmission cannot be interrupted, i.e., service migration only occurs during the processing procedure.

TABLE 1
Summation of Main Notations

Notations	Description
N	The number of MUs
K	The number of PEC servers
$\boldsymbol{\xi}$	The set of service placement configurations
F_k	The computation capability of PEC server k
B_k	Total available bandwidth of PEC server k
D_k	Maximum storable data size of PEC server k
d_i	Service data size of MU i
h_k	The CPU utilization efficiency of PEC server k
n_k	The number of MUs that occupy PEC server k
$E_i(t)$	The service migration cost of MU i

PEC server k can be specified as a tuple $\{F_k, B_k, D_k\}$, known as Sever State Information (SSI). Variable F_k denotes the computation capability, and B_k is the total available bandwidth. Variable D_k is the maximum data size that server k can storage. To preserve the privacy of MUs, PEC servers only broadcast SSI while the detailed information about service requests is confidential. Herein, service placement can be denoted by configuration ξ. For any available $\xi \in \boldsymbol{\xi}$, it can be viewed as a determined strategy profile of PEC resource allocation. Let $\xi(t)$ denote the configuration at time slot t, represented by:

$$\xi(t) = \begin{bmatrix} \xi_1^1(t) & \xi_1^2(t) & \ldots & \xi_1^K(t) \\ \xi_2^1(t) & \xi_2^2(t) & \ldots & \xi_2^K(t) \\ \vdots & \vdots & \ddots & \vdots \\ \xi_N^1(t) & \xi_N^2(t) & \ldots & \xi_N^K(t) \end{bmatrix} \in \mathbb{R}^{N \times K},$$

where $\xi_i^k(t) \in \{0, 1\}$. Equation $\xi_i^k(t) = 1$ denotes PEC server k provides services for MU i at time slot t, otherwise $\xi_i^k(t) = 0$. Note that each MU can merely be served by one PEC server within a time slot. For simplicity of illustration, notations $\xi(t)$ and ξ as well as $\xi(t+1)$ and ξ' are inter-changeable in this paper. In the following, the system utility function is defined based on the service execution latency (including uploading, processing and backhaul latency) and CPU utilization efficiency of PEC servers. Main notations are summarized in Table 1.

3.1 Service Execution Utility

Traditional single-tenancy (i.e., service requests are cached in a queue and processed sequentially) limits the CPU utilization efficiency of PEC servers. It also incurs additional queuing latency, leading to unsatisfactory services under the circumstance of numerous MUs. Therefore, multi-tenancy is leveraged for high CPU utilization efficiency and low service execution latency. In our constructed PEC-enabled stochastic mobility model, MUs that occupy the same PEC server share resources equally. By this scheduling, computation resources of PEC servers can be efficiently and fairly utilized. This model can be easily extended to a generalized one by assigning different priorities to MUs.

Based on the experimental results illustrated in [32], the CPU utilization efficiency for PEC server k can be approximated by:

$$h_k(\xi, t) = -\log_{a_k}(n_k(\xi, t)), \tag{1}$$

where parameter $a_k \in (0.9, 1.0)$ is determined by the data size of service request. Variable $n_k(\xi)$ denotes the number of MUs that occupy PEC server k according to configuration ξ, which can be computed by:

$$n_k(\xi, t) = \sum_{i\in\mathcal{N}} \xi_i^k(t). \tag{2}$$

Generally, providing services for more MUs enables higher CPU utilization efficiency. Thus, $h_k(\xi)$ increases with the rising number of MUs that share resources of server k. However, CPU utilization efficiency can converge to a certain value even variable $n_k(\xi)$ continues to increase. Considering the limitation on computation and storage resources of PEC servers, we set the maximum storage capacity of each server as D_k [33].

There are three phases of service execution for each MU i, including uploading $\tau_i^u \rightarrow \tau_i^p$, processing $\tau_i^p \rightarrow \tau_i^b$ and backhaul $\tau_i^b \rightarrow \tau_i^e$, where τ_i^u denotes the time when MU i generates the service request. Then, MU i uploads the service data to the selected PEC server, and the uploading procedure completes at time slot τ_i^p, which can be also viewed as the time when the selected PEC server receives the service data and starts to process. During the period of $\tau_i^p \rightarrow \tau_i^b$, the service request is processed and migrated among PEC servers. At time slot τ_i^b, PEC server k' finishes request processing and sends results back to MU i. The backhaul transmission completes at time slot τ_i^e. For the uploading procedure, data size d_i satisfies:

$$d_i = \sum_{t=\tau_i^u}^{\tau_i^p} B_k g_{i,k}(t)\Delta t, \tag{3}$$

where time-dependent variable $g_{i,k}(t)$ denotes the communication channel gain between MU i and PEC server k, computed by:

$$g_{i,k}(t) = l_{i,k}^{-z}(t). \tag{4}$$

Variable $l_{i,k}(t)$ denotes the distance between MU i and PEC server k at time slot t, and coefficient $z \in [2, 4]$. Since MUs communicate with PEC servers through OFDMA, the Signal to Noise Ratio (SNR) between MU i and PEC server k can be calculated by:

$$SNR_{i,k}(t) = \frac{p_i g_{i,k}(t)}{\sigma^2}, \tag{5}$$

where p_i and σ^2 denote the transmission power of MU i and noise. Both of them are time-independent and relatively fixed during network access. Thus, the SNR is numerically proportional to the channel gain, i.e., $SNR_{i,k}(t) \propto g_{i,k}(t)$.

According to Shannon equation, the transmission rate between MU i and server k is $r_{i,k}(t) = B\log_2(\mathrm{SNR}_{i,k}(t))$, and logarithmic function with base 2 is monotonically increasing. To simplify the investigated problem properly, we define transmission rate function $r_{i,k}(t)$ as a linear function with respect to the channel bandwidth B and the channel gain approximated by equation (4), and ignore the constant coefficient P_i/σ^2 as well as logarithmic function $\log_2(\mathrm{SNR}_{i,k}(t))$. Then, $r_{i,k}(t)$ can be represented by: $r_{i,k}(t) \triangleq Bg_{i,k}(t)$. The user-dependent variable channel gain is retained. For each user, the same operation is done on the channel gain when calculating the transmission rate. Thus, constant coefficients can be ignored without causing large errors.

It can be observed that the uploading latency, denoted by $|\tau_i^p - \tau_i^u|$, depends on the transmitted data size, allocated bandwidth of PEC servers and the distance between MUs and PEC servers. Denote the transmission power of MU i by p_i, and the uploading cost of MU i can be calculated by:

$$U_i^{\tau_{u\rightarrow p}} = p_i|\tau_i^p - \tau_i^u|. \tag{6}$$

After data transmission is finished at time slot τ_i^p, PEC server k starts to process the request. During time slot t, the amount of computation task that PEC server k completes for MU i can be computed by:

$$c_{i,k}^t(\xi, t) = \frac{F_k}{n_k(\xi, t)}\Delta t. \tag{7}$$

Then, the remaining task to be completed in the future time slots can be represented by $\max\{C_i - c_{i,k}^t(\xi, t), 0\}$. Given the service request of MU i accomplished at time slot τ_i^b, the following equation holds:

$$C_i = c_{i,k}^{\tau_i^p}(\xi, t) + \sum_{t'=\tau_i^p+1}^{\tau_i^b} c_{i,k'}^{t'}(\xi', t), \tag{8}$$

where PEC server $k' \in \mathcal{K}$ and $k' \neq k$, as long as MU i moves out of the wireless coverage of server k. Correspondingly, the processing cost of PEC server k can be computed by:

$$U_i^{\tau_{p\rightarrow b}} = \sum_{t=\tau_i^p}^{\tau_i^b}\sum_{k=1}^{K} \xi_{i,k}(t) p_k \Delta t. \tag{9}$$

If PEC server $\widetilde{k}$ accomplishes the request processing of MU i at time slot τ_i^b, the service results are sent back to MU i during time slots τ_i^b to τ_i^e. If the data size of results equals to the input data size d_i, the following backhaul transmission constraint holds:

$$d_i = \sum_{t=\tau_i^b}^{\tau_i^e} B_{\widetilde{k}} g_{i,\widetilde{k}}(t)\Delta t. \tag{10}$$

Similar to the uploading model in equation (6), the backhaul cost of server k^- can be calculated by:

$$U_i^{\tau_{b\rightarrow e}} = p_{k^-}|\tau_i^e - \tau_i^b|. \tag{11}$$

Generally, local computing is considered in two situations. In partial offloading, each task can be divided into subtasks. Local computing and MEC cooperate to accomplish the whole task. In full offloading, each task is either offloaded to a MEC server or processed locally. Two main reasons motivate us to encourage PEC and ignore local computing in this paper: First, our framework is oriented towards delay-sensitive applications. It is difficult for mobile devices, such as mobile phones and laptops, to process these complicated tasks in time, which also promotes the development of MEC. Fully utilizing PEC resources is able to not only save local energy, but also improve users' quality of service; Second, UAVs can be deployed to provide agile computing services for urgent tasks, which are atomic and cannot be further divided. For example, Internet of UAVs can provide remote health monitoring services for patients infected with COVID-19. The medical analysis task is complicated and time-sensitive. In addition, the monitored data are atomic and need to be transmitted as a whole. Thus, we focus on service placement at the PEC server and ignore local computing.

3.2 Service Migration Cost

During the period of service processing, MUs move erratically across the coverage of PEC servers. By dynamically migrating service instances across servers to follow the mobility of MUs, the stability of communication can be guaranteed, and the service execution latency can be significantly reduced. With service migration, MUs and PEC servers can be connected based on single-hop communication[1], which can enhance the robustness of the offloading framework and avoid backhaul failure and transmission timeout caused by multi-hop communication. In time slot $t+1$, the service migration cost of MU i can be calculated by:

$$E_i(t+1) = \sum_{k=1}^{K}\sum_{k'=1}^{K} \xi_{i,k}(t)\xi_{i,k'}(t+1)E^i_{k,k'}, \tag{12}$$

where $E^i_{k,k'}$ denotes the service migration cost from PEC servers k to k' for MU i. When MU i moves out of the wireless coverage of server k, it selects a new PEC server k' that can maximize the system utility, by updating its service placement configuration ξ_i. Then, PEC server k migrates the unfinished service request of MU i to server k'. The corresponding migration cost can be defined as:

$$E^i_{k,k'} = \begin{cases} \rho d_i s_{k,k'}, & \text{if } k \neq k', \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

where $s_{k,k'}$ denotes the Euclidean distance between servers k and k', and variable ρ is the coefficient.

Most existing researches consider the semi-dynamic model and assume service requests can be completed within a time slot, before MUs leave the wireless coverage of the corresponding MEC server [8], [10]. In contrast, we construct a dynamic framework that service requests can last for a few time slots for execution. During this period, MUs can move erratically across the coverage of MEC servers. It is worth noting that the backhaul may fail without service migration, and it is difficult to acquire global information in such a PEC environment. To conquer the above challenges, we proposed DASS algorithm to configure the service placement profile, which can efficiently realize service migration.

1. Single-hop communication here refers to the communication without the assist of any other intermediate PEC servers.

3.3 Problem Formulation

By integrating the service execution utility and migration cost, we can obtain the utility of MU i for the whole service execution procedure by:

$$U_i = \beta(U_i^{\tau_{u\to p}} + U_i^{\tau_{p\to b}} + U_i^{\tau_{b\to e}}) + \gamma \sum_{t=\tau_i^p+1}^{\tau_i^b} E_i(t). \tag{14}$$

With the objective of maximizing the system utility, the stochastic process optimization problem can be formulated by:

$$\mathbf{UM}: \max_{\xi\in\xi} \quad \alpha\sum_{k\in\mathcal{K}} h_k(\xi,t) - \sum_{i\in\mathcal{N}} U_i,$$

s.t.

$$\sum_{i=1}^{N} \xi_{i,k}(t)d_i \leqslant D_k, \forall k\in\mathcal{K}, \tag{$\mathbf{UM}a$}$$

$$|\tau_i^e - \tau_i^u| \leqslant T_i^{max}, \forall i\in\mathcal{N}, \tag{$\mathbf{UM}b$}$$

$$\xi_{i,k}\in\{0,1\}, \forall i\in\mathcal{N}, k\in\mathcal{K}, \tag{$\mathbf{UM}c$}$$

$$\text{Equations (3), (8), (10),}$$

where **UM** is short for the utility maximization problem. Constraint (**UM**a) requires that the loaded data size of PEC server k cannot exceed its maximum storage capacity. Constraint (**UM**b) guarantees that the service request can be accomplished within the tolerable latency. Constraint (**UM**c) illustrates that the service request of each MU can be merely allocated to one PEC server in each time slot. Equations (3), (8) and (10) determine time slots τ_i^p, τ_i^b and τ_i^e, respectively. The optimization problem **UM** is complicated for three reasons:

- Theoretically, the optional space of the service placement configuration $|\xi|$ in each time slot equals to K^N. Based on exhaustive search, the long-term searching space is $|\tau_i^e - \tau_i^u|\cdot K^N$, which is extremely large. Since the service requirements of MUs are usually latency-sensitive, traditional methods are not efficient.
- Previous service placement configuration strategies can influence consecutive service placement decisions. In addition, the centralized optimal solution requires the global mobility information of all MUs, which is difficult to obtain. Thus, it is significant to develop a distributed and dynamic algorithm to approximate the optimal service placement without a centralized controller.
- MUs do not generate service requests synchronously in one time slot. In contrast, the generation, processing and data transmission of service requests are asynchronous in reality. Thus, it is difficult to obtain

$$\mathcal{U}_i(t) = \begin{cases} \beta(p_i|\tau_i^p - \tau_i^u| + \sum_{t=\tau_i^p}^{\tau_i^b} \sum_{k=1}^{K} \xi_{i,k}(t) p_k \Delta t + p_{k^-}|\tau_i^e - \tau_i^b|) + \gamma \sum_{t=\tau_i^p+1}^{\tau_i^b} E_i(t), & t \in [\tau_i^u, \tau_i^p], \\ \beta(\sum_{t'=t}^{\tau_i^b} \sum_{k=1}^{K} \xi_{i,k}(t') p_k \Delta t + p_{k^-}|\tau_i^e - \tau_i^b|) + \gamma \sum_{t'=t+1}^{\tau_i^b} E_i(t'), & t \in (\tau_i^p, \tau_i^b], \\ \beta(p_{k^-}|\tau_i^e - \tau_i^b|), & t \in (\tau_i^b, \tau_i^e]. \end{cases} \tag{15}$$

a centralized method for service migration in the PEC-enabled system.

Before developing online dynamic algorithm, the utility function of MU i is reformulated based on three periods of service execution, as shown in equation (15). Note that the uploading utility of MU i, denoted by $p_i|\tau_i^p - \tau_i^u|$, is fixed as a constant rather than variable $p_i|\tau_i^p - t|$ when $t \in [\tau_i^u, \tau_i^p]$. This is because service migration merely occurs during request processing. The service placement configuration only needs to be decided at the beginning of the service request generation period, i.e., $t = \tau_i^u$. For $t \in (\tau_i^u, \tau_i^p]$, the service placement configuration remains unchanged, i.e., $\xi_i(t) = \xi_i(\tau_i^u), \forall t \in [\tau_i^u, \tau_i^p]$. Similarly, the backhaul transmission utility is also fixed as $p_{k^-}|\tau_i^e - \tau_i^b|$.

In our model, deriving a long-term solution is unrealistic since it requires the global future information of all MUs. With the objective of maximizing the system utility in each time slot, the optimization problem can be reformulated by:

$$\mathbf{Online-UM}: \max_{\xi \in \boldsymbol{\xi}} \quad \alpha \sum_{k \in \mathcal{K}} h_k(\xi, t) - \sum_{i \in \mathcal{N}} \mathcal{U}_i(t), \forall t \in \mathcal{T},$$

s.t.

$$\sum_{i=1}^{N} \xi_{i,k}(t) d_i \leq D_k, \forall k \in \mathcal{K}, \tag{$\mathbf{Online-UM}a$}$$

$$|\tau_i^e - t| \leq |T_i^{max} - t + \tau_i^u|, \forall i \in \mathcal{N}, \tag{$\mathbf{Online-UM}b$}$$

$$\xi_{i,k} \in \{0, 1\}, \forall i \in \mathcal{N}, k \in \mathcal{K}, \tag{$\mathbf{Online-UM}c$}$$

$$\text{Equations } (3), (8), (10).$$

There are two main differences between problem **UM** and **Online-UM**. First, the utility function of each MU in **UM** is a long-term utility, including those related to uploading, processing and backhaul procedures. However, the utility function in **Online-UM** represents the utility of each time slot. Since previous service placement configuration can influence consecutive service placement decisions, the utility of MU i in time slot t cannot only be impacted by current decisions, but also further ones. Second, the service execution latency starts at time slot t, and the latency threshold needs to minus the already consumed time, i.e., $|t - \tau_i^u|$. The corresponding constraint in (**Online-UM**b) is updated.

4 Online Dynamic Algorithm

In this section, a novel dynamic framework is developed for online service placement. Three processes are included: First, the storage queue of PEC servers is optimized by Lyapunov optimization method; Then, SAA algorithm [20] is leveraged for future expected utility approximation; Finally, the utility maximization problem is solved by utilizing Markov approximation technique.

4.1 Lyapunov Optimization for Queue Stability

In this paper, Lyapunov optimization algorithm is utilized to stabilize the storage queue of PEC servers. With this objective, matching algorithm [34] and game theory [17] can be alternatives. However, matching algorithm is centralized, and game theory requires frequent information change to update policy iteratively, which is time-consuming. In order to develop an online distributed method, Lyapunov optimization is a good choice.

In problem **Online-UM**, the long-term storage constraints of all PEC servers, i.e., equation (**Online-UM**a), make the service placement decisions coupled across different time slots. Besides, the system utility includes both the utilities of PEC servers and those of MUs, whose intrinsic relevance makes the problem difficult to decouple. To address the above challenges, we utilize the Lyapunov optimization method [35] to guarantee that service placement configurations meet the long-term storage constraints. By introducing a virtual queue, Lyapunov optimization is able to balance the trade-off between queue stability and utility maximization. In our framework, the time-dependent service request backlog of PEC server k can be formulated as a dynamic backlog queue, represented by:

$$Q_k(t+1) = \max\{Q_k(t) + \Delta D_k(t) - D_k, 0\}, \tag{16}$$

where queue length $Q_k(t)$ denotes the exceeded load on server k in time slot t. Variable $\Delta D_k(t)$ represents the throughput of PEC server k, which can be calculated by $\Delta D_k(t) = D_k^{in}(t) - D_k^{out}(t)$. Intuitively, the value of $Q_k(t)$ indicates the deviation of current backlog from the storage threshold of server k. A positive value of $Q_k(t)$ implies the backlog exceeds the storage threshold. Given $Q_k(0) = 0$, it can be derived that the expected backlog during all time slots is less than the threshold [22]:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[Q_k(t)] \leq \lim_{T \to \infty} \frac{\mathbb{E}[Q_k(T)]}{T} + D_k. \tag{17}$$

Hence, constraint (**Online-UM**a) can be guaranteed by stabilizing backlog queue $Q_k(t)$. With this objective, a quadratic Lyapunov function is defined as follows:

$$L_k(\xi(t)) \triangleq \frac{1}{2} Q_k^2(t), \tag{18}$$

The quadratic Lyapunov function can be regarded as a scalar measure of queue deviation. Similar to $Q_k(t)$, a large value of $L_k(\xi(t))$ (close to 0) implies the storage backlog is overloaded. If we simply minimize the quadratic Lyapunov function to its lower bound, i.e., 0, the storage constraint can be met. However, under this configuration, PEC servers do not provide any services for MUs since all requests are refused, which is contrary to our original intention. Therefore, a one-step conditional Lyapunov drift function is introduced:

$$\Delta_k(t) \triangleq \mathbb{E}[L_k(\xi(t+1)) - L_k(\xi(t)) | \xi(t)]. \tag{19}$$

The above function records the change of the backlog queue over one time slot. In this case, the long-term constraint (**Online-UM**a) in problem **Online-UM** can be decomposed into a series of queue stability constraints in each time slot. In order to maximize the system utility under the premise of queue stability, a Lyapunov drift-plus-penalty function is defined by:

$$\Delta_k(t) + VP_k(\xi, t), \tag{20}$$

where V is a non-negative weight coefficient that balances the trade-off between the backlog of service request queue and the utility. Variable $P_k(t)$ is the PEC server-oriented cost function, which can be computed by:

$$P_k(\xi, t) = \sum_{i=1}^{N} \xi_{i,k}(t)\mathcal{U}_i(t) - \gamma^h h_k(\xi, t). \tag{21}$$

Note that the summation of PEC server-oriented cost equals to the opposite of the utility in problem **Online-UM**. Thus, the system utility can be maximized by minimizing the PEC server-oriented cost. Moreover, minimizing the Lyapunov drift function, i.e., equation (19), is equivalent to minimize the change of backlog and stabilize the backlog queue over different time slots. Therefore, by minimizing the Lyapunov drift-plus-penalty, the system utility can be maximized while the length of the backlog queue is towards the storage threshold. Since the request arrival rate of each queue is bounded, positive upper bound B_k should be guaranteed by the following property:

$$\begin{aligned} E[B_k(t)|Q_k(t)] &= \frac{1}{2}\Big(\mathbb{E}[\Delta D_k(t)]^2 + D_k^2 - 2\mathbb{E}[\Delta D_k(t)]D_k\Big) \\ &\le B_k. \end{aligned} \tag{22}$$

In addition, the following lemma provides the supermum of the Lyapunov drift-plus-penalty function.

Lemma 1. *Given a small real positive number $\epsilon > 0$:*

$$\mathbb{E}[\Delta D_k(t) - D_k|Q(t)] \le -\epsilon,$$

the following property holds:

$$\mathbb{E}[\Delta_k(t) + VP_k(t)|Q_k(t)] \le B_k + VP_k - \epsilon Q_k(t). \tag{23}$$

The proof can be found in *Appendix A*, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3046000.

Following **Lemma 1**, it can be observed that minimizing the Lyapunov drift-plus-penalty is equivalent to minimizing the right term of equation (23). With the objective of minimizing the supermum of the drift-plus-penalty, problem **Online-UM** can be converted to a series of real-time Lyapunov optimization problems. Define the integrated PEC-server oriented utility function by:

$$\mathcal{P}_k(\xi, t) = \epsilon Q_k(t) - VP_k(\xi, t) - B_k. \tag{24}$$

The Lyapunov optimization problem can be formulated as:

$$\mathbf{Lya-UM}: \max_{\xi \in \xi} \quad \sum_{k \in \mathcal{K}} \mathcal{P}_k(\xi, t), \forall t \in \mathcal{T},$$

s.t.

$$|\tau_i^e - t| \le |T_i^{max} - t + \tau_i^u|, \forall i \in \mathcal{N}, \tag{$\mathbf{Lya-UM}a$}$$

$$\xi_{i,k} \in \{0, 1\}, \forall i \in \mathcal{N}, k \in \mathcal{K}, \tag{$\mathbf{Lya-UM}b$}$$

$$\text{Equations } (3), (8), (10).$$

Before solving the above **Lya-UM** problem, we relax constraints (3), (8), (10) and (**Lya-UM**a) by leveraging SAA algorithm [20] to approximate the future utility.

4.2 SAA Algorithm for Future Utility Approximation

In this paper, SAA algorithm is utilized to approximate the future utility by mobility prediction. Many existing researches have focused on mobility prediction. Based on speed-aware framework, the studies in [36] and [37] employ Markov model to predict user's mobility. The authors in [38] predict mobility patterns based on neighbor and destination locations. The above methods require clock synchronization between users and servers. To reduce the communication cost and speed up algorithm convergence, Monte-Carlo based SAA algorithm is selected to improve prediction efficiency at the cost of certain accuracy.

To compute the value of PEC server-oriented utility, three time instants need to be obtained, i.e., τ_i^p, τ_i^b and τ_i^e, which are determined by equations (3), (8) and (10), respectively. However, there are three main challenges to calculate them. First, communication channel gain $g_{i,k}(t)$ is distance-dependent. Considering the stochastic mobility of MUs, the communication distance is time-varying and difficult to obtain; Second, from equation (7), we can observe that request processing time $n_k(\xi)$ is related to the number of MUs that occupy the same PEC server. Variable $n_k(\xi)$ is time-varying, time instant τ_i^b (the time when the PEC server accomplishes the service request) is difficult to estimate; Third, the mobility of MUs is disordered and irregular. Any empirical distribution may deviate from real-word traces, resulting in performance loss. To address these challenges, we utilize SAA algorithm to approximate the expected PEC server-oriented utility.

SAA is a Monte-Carlo based algorithm, which is often utilized to solve multi-slot stochastic problems. By sampling a large number of stochastic scenarios, SAA algorithm approximates the future expected utility for all MUs, and relaxes the constraints of service execution time in problem **Lya-UM**. The details are illustrated in Algorithm 1.

As shown in Fig. 2, MUs move from points S to D during time slot t. Since the distance between MUs and PEC servers (e.g., line $S'O$, $S' \in SD$ in Fig. 2a) varies continuously with the mobility of MUs, it is difficult to compute uploading and backhaul transmission utilities directly by equation (4). Denote the distance between any two points A and B by $|AB|$ (e.g., the distance that MU u_1 moves during time slot t is denoted by $|SD|$). Under the premise that points S, D and O form a triangle, we consider two mobility scenarios as follows:

First, if angle SDO (point D as the vertex) is acute, for any point $S' \in \overline{SD}$, $|OS'| \le \max\{|OS|, |OD|\}$ holds. Hence, we utilize the upper bound of the communication distance

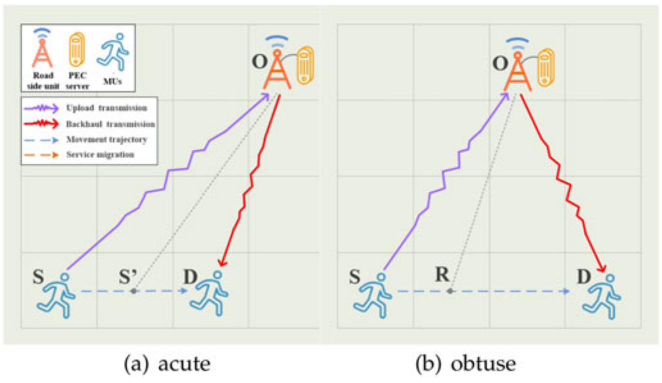


Fig. 2. Two scenarios of communication distance approximation.

during time slot t to approximate the average distance, i.e.,

$$\bar{l}_{i,k}(t) = \max\{|OS|, |OD|\}. \tag{25}$$

Second, if angle SDO is obtuse, we can find a certain point R that $|OR| = \min\{|OS|, |OD|\}$. Without loss of generality, we assume $|OS| < |OD|$, which can be easily extended to the scenario when $|OS| \geq |OD|$. For any point $S' \in \vec{SR}$, $|OS| \geq |OS'|$ holds. Correspondingly, for any point $D' \in \vec{RD}$, $|OD| \geq |OD'|$ holds. Then, the average distance between an MU and the accessed PEC server during time slot t can be approximated by:

$$\bar{l}_{i,k}(t) = \frac{|SR|}{|SD|}|OS| + \frac{|RD|}{|SD|}|OD|. \tag{26}$$

Based on equations (25) and (26), the average channel gain can be calculated. Given a service placement configuration, the long-term system utility can be approximated by accumulating the utilities of all MUs and PEC servers. In order to obtain the *sample optimal* configuration for each time slot, PEC servers need to solve a deterministic equivalent problem, formulated by:

$$\mathbf{DE-UM}: \max_{\xi\in\boldsymbol{\xi}} \quad \sum_{k\in\mathcal{K}} \mathcal{P}_k(\xi, t), \forall t \in \mathcal{T},$$

s.t.

$$\xi_{i,k} \in \{0,1\}, \forall i \in \mathcal{N}, k \in \mathcal{K}. \tag{$\mathbf{DE-UM}a$}$$

In each time slot, SAA algorithm generates a large number of further scenarios (i.e., possible mobility patterns of MUs) based on current time slot. For each scenario, the movement trajectories of MUs are known to PEC servers. Based on the obtained trace, constraints (3), (8) and (10) can be solved by relaxing the calculation of channel gain according to equations (25) and (26). Then, the deterministic equivalent problem becomes a linear programming and is easy to solve. Finally, the future utility can be obtained. The key idea of SAA algorithm is Monte-Carlo based sampling. We do not aim to obtain the accurate future utility, instead, we use Monte-Carlo based sampling to approximate the future utility. In addition, as illustrated in Fig. 3, the number of optional trajectories is small in reality. Thus, SAA algorithm is time-efficient in such cases.

After that, we can obtain several *sample optimal* configurations. In traditional SAA algorithm, these *sample optimal* solutions are evaluated by generating another scenario, and the configuration with the best evaluating performance is chosen as the final optimal solution. However, the contingency of the evaluating sample can lead to a great variance on the performance of SAA algorithm. Thus, we only utilize SAA algorithm to approximate the expected system utility. In the following, Markov approximation is leveraged for the final optimal configuration.

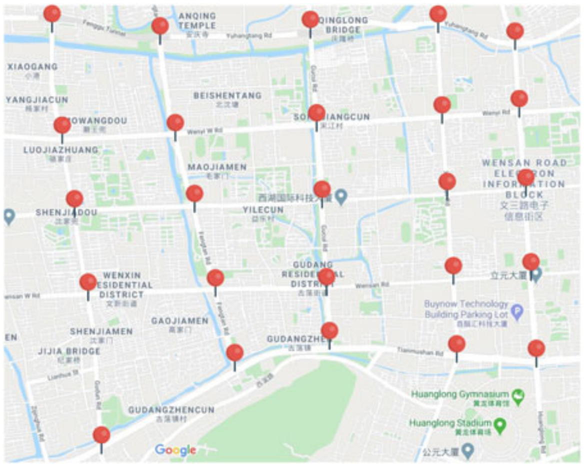


Fig. 3. PEC server deployment in Xiacheng district, Hangzhou, China.

Algorithm 1. SAA-Based Stochastic Utility Approximation Algorithm

```
Generate a sufficient large number of scenarios.
for each scenario do
    MUs specify thier movement trajectories to PEC servers.
    for each PEC server do
        Solve constraints (3), (8) and (10).
        Solve the deterministic equivalent problem DE-UM.
    end
    Return the average system utility as approximated expected
    system utility.
end
```

4.3 Markov Approximation for Dynamic Service Placement

Problem **Lya-UM** is a combinatorial optimization problem, where service placement configuration ξ is a discrete binary decision variable, making the problem complicated to solve. In this section, we utilize Markov approximation algorithm [39] to solve the problem by introducing a continuous probability model. Denote the probability of adopting configuration ξ at time slot t by p_ξ, and the Markov approximation optimization problem can be formulated by:

$$\mathbf{MA-UM}: \max_{\boldsymbol{p}\geq 0} \quad \sum_{\xi\in\boldsymbol{\xi}} p_\xi \sum_{k\in\mathcal{K}} \mathcal{P}_k(\xi, t), \forall t \in \mathcal{T},$$

s.t.

$$\sum_{\xi\in\boldsymbol{\xi}} p_\xi = 1, \tag{$\mathbf{MA-UM}a$}$$

$$\xi_{i,k} \in \{0,1\}, \forall i \in \mathcal{N}, k \in \mathcal{K}. \tag{$\mathbf{MA-UM}b$}$$

The objective of problem **MA-UM** is to find the optimal configuration that can maximize the weighted system utility, which has the same optimal solution as problem **Lya-UM**. $\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)$ can be regarded as the weight of p_ξ.

We utilize convex log-sum-exp function $J_\beta(\xi)$ [40] to approximate the system utility, which is defined as follows:

$$J_\beta(\xi,t)\triangleq\frac{1}{\beta}\log\left(\sum_{\xi\in\boldsymbol{\xi}}\exp\left(\beta\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)\right)\right), \tag{27}$$

where β is a positive constant.

Lemma 2. *The convex log-sum-exp function $J_\beta(\xi)$ can approximate the optimization objective, i.e.,*

$$\max_{\xi\in\boldsymbol{\xi}}\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)\le J_\beta(t)\le\max_{\xi\in\boldsymbol{\xi}}\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)+\frac{1}{\beta}\log|\boldsymbol{\xi}|.$$

The corresponding approximation gap is upper-bounded by $\frac{1}{\beta}\log|\boldsymbol{\xi}|$.

The proof can be found in *Appendix B*, available in the online supplemental material.

Based on **Lemma 2**, it can be yielded that the value of log-sum-exp function $J_\beta(t)$ equals to the optimal solution of the following problem:

$$\mathbf{MA-EN}:\max_{\boldsymbol{p}\ge 0}\quad\sum_{\xi\in\boldsymbol{\xi}}p_\xi\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)-\frac{1}{\beta}\sum_{\xi\in\boldsymbol{\xi}}p_\xi\log p_\xi,\forall t\in\mathcal{T},$$

s.t.

$$\sum_{\xi\in\boldsymbol{\xi}}p_\xi=1, \tag{$\mathbf{MA-EN}a$}$$

$$\xi_{i,k}\in\{0,1\},\forall i\in\mathcal{N},k\in\mathcal{K}. \tag{$\mathbf{MA-EN}b$}$$

By solving the Karush–Kuhn–Tucker (KKT) conditions [40] of problem **MA-EN**, it can be represented by:

$$\begin{aligned}&\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)-\frac{1}{\beta}\log p^*_\xi-\frac{1}{\beta}+\lambda=0,\forall\xi\in\boldsymbol{\xi},\\&\sum_{\xi\in\boldsymbol{\xi}}p^*_\xi(t)=1,\\&\lambda\ge 0,\end{aligned} \tag{28}$$

and the optimal configuration distribution can be derived by:

$$p^*_\xi(t)=\frac{\exp\left(\beta\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)\right)}{\sum_{\xi'\in\boldsymbol{\xi}}\exp\left(\beta\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi',t)(t)\right)},\forall\xi\in\boldsymbol{\xi}. \tag{29}$$

We can observe that the adopting probabilities of different configurations are proportional to their weights (system utilities). In the following, a Markov chain is designed to solve problem **MA-EN** in a distributed way.

The basic idea of distributed Markov approximation is to design a time-reversible Markov chain with stationary distribution $p^*_\xi(t),\xi\in\boldsymbol{\xi}$. Denote the transition probability between two configurations ξ and $\xi^{'}$ by $q_{\xi,\xi^{'}}$. If configuration $\xi^{'}$ can be reached by modifying the service placement decision of only one MU in configuration ξ, we call that the two configurations are directly connected (otherwise, undirected connected) in the Markov chain. Let the transition probability of all undirected connections be 0, i.e., removing transition edges among those states[2] that are not adjacent in the Markov chain. It has been illustrated that the modified Markov chain is still time-reversible in [39], since all states are still reachable. For two directly connected states, the transition probability is defined to be negatively correlated with the system utility under configuration ξ, represented by:

$$q_{\xi,\xi^{'}}=\alpha\left[\exp\left(\beta\sum_{k\in\mathcal{K}}\mathcal{P}_k(\xi,t)\right)\right]^{-1}, \tag{30}$$

where α is a positive constant. Correspondingly, $q_{\xi^{'},\xi}$ is defined symmetrically. In particular, based on the transition probability matrix in equation (30), we can obtain a mixed strategy, i.e., the stability point can be uniquely determined by the mixed strategy. Thus, we do not make any assumption regarding multiple stability points (also known as multiple pure strategies).

In general, the convergence rate of Markov chains mainly depends on the number of optional states. In this paper, without any constraints, the optional state of the configuration variable $\xi(t)$ is N^K for each time slot, where N and K denote the number of MUs and PEC servers, respectively. Indeed, N^K increases explosively in dense dynamic networks, making it difficult for the Markov chain to converge. However, two real-world constraints on both N and K guarantee the efficiency of our proposed algorithm in dense networks: 1) In reality, limited number of PEC servers can be accessed by one user in each time slot, and the optional space of K is small; 2) Service requests are generated by following Bernoulli process, indicating that PEC services are not required by all MUs in each time slot. With the storage capability constraint of PEC servers formulated in (**UM**a), limited MUs can be served by one server.

Theorem 1. *With the above definition of transition probabilities, the Markov chain follows two properties:*

1) *The Markov chain is irreducible, and any two states are reachable directly or through other states.*
2) *Given any two different states ξ and $\xi^{'}$, the balance equation of stationary distribution is satisfied, i.e.,*

$$p^*_\xi(t)q_{\xi,\xi^{'}}=p^*_{\xi^{'}}(t)q_{\xi^{'},\xi}.$$

The proof can be found in *Appendix C*, available in the online supplemental material.

The dynamic storage-stable service placement algorithm is illustrated in Algorithm 2. For each time slot, PEC servers construct a time-reversible Markov chain, and the probability distribution of the service placement configuration is initialized randomly. Then, SAA algorithm is utilized to approximate the future expected utility by solving constraints (3), (8) and (10). After that, PEC servers can calculate the system utility and the transition probability according to equation (30). Based on **Theorem 1**, one state can reach any state within finite transitions. Hence, the Markov chain is

2. Each optional configuration is a state in Markov chain. Here the configuration and state are interchangeable.

guaranteed to reach stationary state after a finite number of iterations.

Algorithm 2. Dynamic Storage-Stable Service Placement Algorithm

```
Initialize the service placement configuration randomly.
for each time slot $t$ do
  Initialize the service placement probability distribution randomly.
  MUs approximate the future expected utiltiy by Algorithm 1.
  for each PEC server do
    Calculate PEC server-oriented utility.
    Construct the transition probability according to (30).
    Derive stationary Markov chain iteratively.
  end
  Update the service placement configuration at time slot $t$ according to (29).
end
```

5 Performance Analysis

In this section, we analyze the performance of the proposed DASS algorithm theoretically. First, the time complexity of DASS algorithm is analyzed. Then, we discuss system performance in terms of the system utility and backlog queues of PEC servers. Specifically, we derive the performance gap between our approximated solution and the theoretical optimal solution. Two upper bounds of backlog queues of PEC servers are yielded to demonstrate the effectiveness of DASS algorithm, which can meet the storage constraint of PEC servers.

Theorem 2. *Denote the average number of iterations for PEC servers to approximate the future utility and obtain the stationary Markov chain by C and I, respectively. Given the number of MUs N and PEC servers K, the time complexity of the proposed DASS algorithm is $O(CNK^2 + INK)$.*

The proof can be found in *Appendix D*, available in the online supplemental material.

Theorem 3. *Given coefficients V and β, the performance gap between our approximated solution and the theoretical optimal solution can be represented by:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k\in\mathcal{K}}\mathbb{E}[P_k(\xi,t)] \le \sum_{k\in\mathcal{K}} P_k^*(\xi,t) + \frac{1}{\beta V}\log|\boldsymbol{\xi}| + \sum_{k\in\mathcal{K}}\left(\frac{B_k}{V} + \frac{\mathbb{E}[L_k(\xi(0))]}{VT}\right).$$

The proof can be found in *Appendix E*, available in the online supplemental material.

Theorem 4. *For each PEC server k, its backlog queue is constrained by a V-independent upper bound, represented by:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[Q_k(t)] \le \frac{B_k}{\epsilon} + \frac{\mathbb{E}[L_k(0)]}{\epsilon T}.$$

For the backlog queues of all PEC servers, there is a V-dependent upper bound, denoted by:

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k\in\mathcal{K}}\mathbb{E}[Q_k(t)] \le \frac{\sum_{k\in\mathcal{K}}\left(P_k^*(\xi,t) - P_{min}\right)}{\epsilon} + \frac{1}{\beta V\epsilon}\log|\boldsymbol{\xi}| + \sum_{k\in\mathcal{K}}\left(\frac{B_k}{V\epsilon} + \frac{\mathbb{E}[L_k(\xi(0))]}{VT\epsilon}\right),$$

where P_{min} denotes the minimum PEC server-oriented utility, i.e.,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[P_k(\xi,t)] \ge P_{min}.$$

The proof can be found in *Appendix F*, available in the online supplemental material.

6 Performance Evaluation

In this section, we conduct two sets of simulations to evaluate our proposed DASS algorithm in terms of several performance indicators.

6.1 Simulation Setup

Scenarios. We simulate the PEC-enabled service placement framework based on two different scenarios. One is the 100×100 grid map shown in Fig. 1, where $K = 20$ PEC servers are deployed to provide services for $N \in [100, 200]$ MUs. During each time slot, MUs move randomly towards up, down, left or right for one unit grid. The wireless coverage radius of PEC servers is set as 30 unit grids, i.e., the communication distance between MUs and PEC servers is $d \in (0, 30]$. The other scenario is based on the real-word traces of vehicles in Hangzhou, China. As shown in Fig. 3, we collect trajectories of more than 300 vehicles in Xiacheng district in October, 2017. We deploy PEC servers at each intersection, with $K = 25$. The average number of vehicles per minute varies throughout the day from 0 to 48.

Parameters. The transmission power of MUs is set as $p_i = 100\ \mathrm{mWatts}, i \in \mathcal{N}$ [41]. The data size of service requests is generated randomly from [40, 140] MB. Generally, the required CPU cycles of service requests are positive correlated to the corresponding data sizes. We set the required CPU cycles $C_i \in$ [6000, 10000] Megacycles. The CPU frequency of PEC servers is 2.4 GHz, and the storage capability is 500 MB [20]. The length of each time slot is equal. In our experiment, the length of one slot is set to 0.1s. However, it is not the only feasible setting. Actually, based on this hyper parameter setting, there is a trade-off between algorithm execution cost and performance accuracy. For example, if the length of one slot is small, the distance of user movement is also small during one time slot. The utility approximation is relatively accurate. However, the algorithm needs to be executed frequently to make the queue of PEC servers stable, resulting in extra cost.

Performance Indicators. 1) The optimization objective of problem **UM**, i.e., average system utility; 2) The number of completed service requests within a certain period of time. We introduce this performance indicator to demonstrate that our proposed DASS algorithm outperforms other schemes in terms of service efficiency and popularity (serve more MUs within unit time); 3) The number of service migrations. In order to reach a satisfactory trade-off between service robustness and system utility, our framework intends to serve as

TABLE 2
Statistics on the Number of Access MUs for Each Server

	Server 1	Server 2	Server 3	Server 4	Server 5
Mode	6	6	5	7	4
Max	14	13	13	15	13
	Server 6	Server 7	Server 8	Server 9	Server 10
Mode	6	7	6	6	6
Max	14	13	14	14	15
	Server 11	Server 12	Server 13	Server 14	Server 15
Mode	6	8	7	8	6
Max	13	12	13	14	12
	Server 16	Server 17	Server 18	Server 19	Server 20
Mode	6	6	5	5	4
Max	12	13	13	12	12

many MUs as possible at the cost of the least number of migrations; 4) Algorithm execution time within a certain episodes. It is required for online algorithm that the execution cycle is short.

Benchmarks. Three methods are leveraged for performance comparison, including Myopic Best Response (MBR) [42], EUAGame [21] and Distance Minimum Principle (DMP) algorithms. MBR algorithm makes service placement decisions based on the instantaneous environments, e.g., service data information and the communication distance between MUs and PEC servers. In other words, it does not consider the resources competition incurred by other individuals. EUAGame approach aims at maximizing the number of served MUs while minimizing the cost of application vendors. It transforms the service placement problem into a variable size bin packing problem, and a Nash equilibrium can be reached through a few iterations. DMP method always chooses the nearest PEC server to offload the service data.

6.2 Numerical Results

We run the proposed DASS algorithm for 20000 time slots in the simulated 100×100 grid map, and count the mode and maximum number of MUs accessed to each PEC server in Table 2. In each time slot, MUs select PEC servers based on the mixed strategy derived by equation (29). The role of mode reflects the number of MUs accessed to each server most of the time, while the role of max indicates the peak number of accessed MUs. We notice that the number of accessed MUs for all servers is generally between 4 and 8. Occasionally, the number of accessed MUs for one server exceeds 10, and its highest value is 15. Constrained by storage capability, PEC servers can be accessed by limited MUs at one time, preventing network overload.

Average System Utility With Different Numbers of MUs and Vehicles: Figs. 4a and 5a compare the average system utility with different number of participants under simulated and real-world scenarios, respectively. With sufficient PEC resources, the average system utility keeps stable with the increasing number of MUs or vehicles. In the simulated scenario (Fig. 4a), DASS algorithm can increase 7, 27 and 57 percent average system utilities compared with EUAGame, MBR and DMP methods, respectively. In the real-world scenario (Fig. 5a), DASS algorithm can improve 5, 31 and 55 percent performance gains compared with EUAGame, MBR and DMP methods, respectively. Our proposed DASS algorithm outperforms other three methods for a few reasons. First, compared with EUAGame method, DASS algorithm takes the future approximated utility into consideration. EUAGame aims at reaching the equilibrium merely based on current states. Then, MBR and DMP methods ignore the mobility of MUs and vehicles, and make local optimal service placement decisions. Numerical results demonstrate that it is necessary for PEC service providers to consider the possible movement trajectories of MUs,

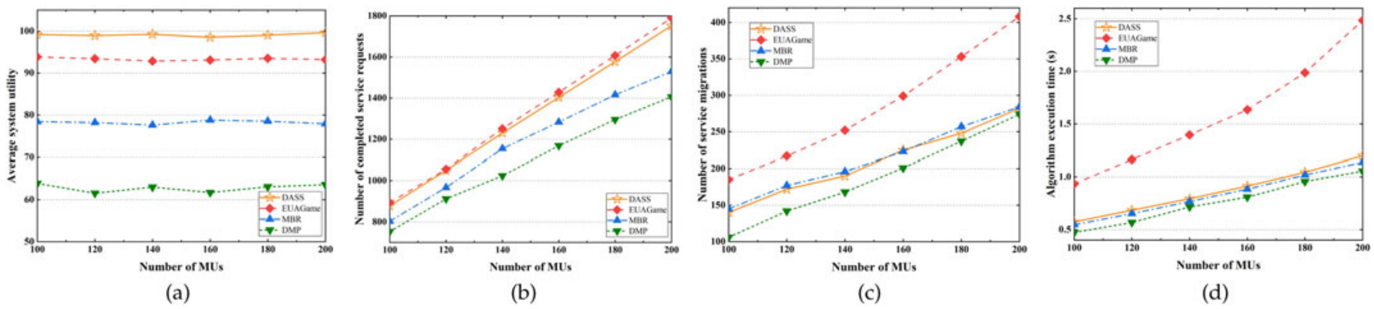


Fig. 4. Performance comparison with different number of MUs under the simulated scenario: (a) Average system utility. (b) Number of completed service requests. (c) Number of service migrations. (d) Algorithm execution time.

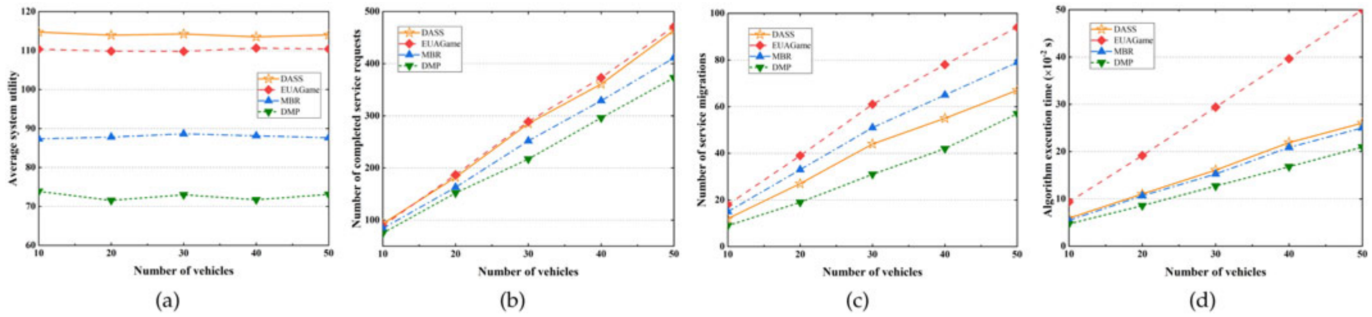


Fig. 5. Performance comparison with different number of vehicles under the real-world scenario: (a) Average system utility. (b) Number of completed service requests. (c) Number of service migrations. (d) Algorithm execution time.

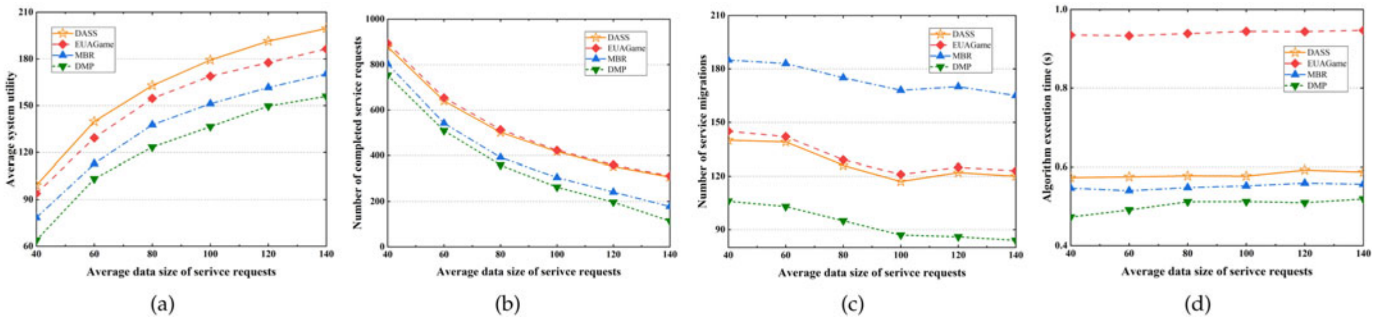


Fig. 6. Performance comparison with different data sizes of service requests under the simulated scenario: (a) Average system utility. (b) Number of completed service requests. (c) Number of service migrations. (d) Algorithm execution time.

especially when service requests cannot be completed by one server and need to be migrated.

The Number of Completed Service Requests With Different Numbers of MUs and Vehicles. Figs. 4b and 5b evaluate the impact caused by the number of MUs and vehicles on the number of completed requests, respectively. It can be observed that the number of completed requests increases with the rising number of MUs and vehicles. Aiming at maximizing the number of completed requests, the performance of EUAGame slightly exceeds that of DASS (5 percent when $N = 160$ in Fig. 4b). However, the performance of the system utility and algorithm execution time (see Figs. 4d and 5d) is worse. In summary, our proposed DASS algorithm is able to benefit MUs as much as possible with a relatively high utility and low time cost.

The Number of Service Migrations With Different Numbers of MUs and Vehicles. Figs. 4c and 5c compare the number of service migrations with different number of MUs and vehicles, respectively. Since DMP method always offloads the service request to the nearest PEC server, the number of service migrations is small. Even though, when MUs or vehicles are not evenly distributed, a few PEC servers can be overloaded while others are idle. It can be observed that the performance gap between EUAGame and DASS algorithms in the real-world scenario is smaller than that in the simulated scenario. This is because roads in the real-word are more regulated than those in simulation (Actually, MUs move erratically in the grid map without any road constraint). The mobility of vehicles is regular and easy to capture. Thus, the number of service migrations for all algorithms tends to be the same. In summary, DASS algorithm does not incur frequent data transmission and avoid much communication cost.

Algorithm Execution Time With Different Numbers of MUs and Vehicles. Figs. 4d and 5d compare algorithm execution time with different number of MUs and vehicles for 100 episodes, respectively. We can observe that DASS and MBR algorithms consume approximately the same time. The time that DASS takes is 10 and 24 percent higher than that of the heuristic DMP algorithm in the simulated and real-world scenarios, respectively. Since the cardinality of the algorithm execution time is low, our proposed DASS algorithm can obtain a better system utility with a tolerable time consumption. In addition, since EUAGame method obtains the optimal service placement decision by an iterative game theory approach, it takes a relatively long time to reach the equilibrium, and its execution time is greatly affected by the number of MUs and vehicles.

Average System Utility With Different Data Sizes of Service Requests. Figs. 6a and 7a show the trends of average system utility with different data size of service requests in the simulated and real-world scenarios, respectively. Since the service reward is positively correlated to the corresponding data size, the average system utility increases with the rising data size of service requests. Compared with EUAGame, MBR and DMP methods, our proposed DASS algorithm can increase the average system utility by 8, 19, and 32 percent in the simulated scenario, respectively. Since the real-world scenario is regulated, the performance gaps among algorithms are narrowed. DASS algorithm can achieve 4, 10, 26 percent performance gains compared with other three algorithms, respectively. This result demonstrates that our proposed DASS can deal with complicated scenarios, and it is robust and extensible.

The Number of Completed Service Requests With Different Data Sizes of Service Requests. In Figs. 6a and 7a, as the data size continues to increase, the growth trend of average system utility gradually slows down. This is because it takes

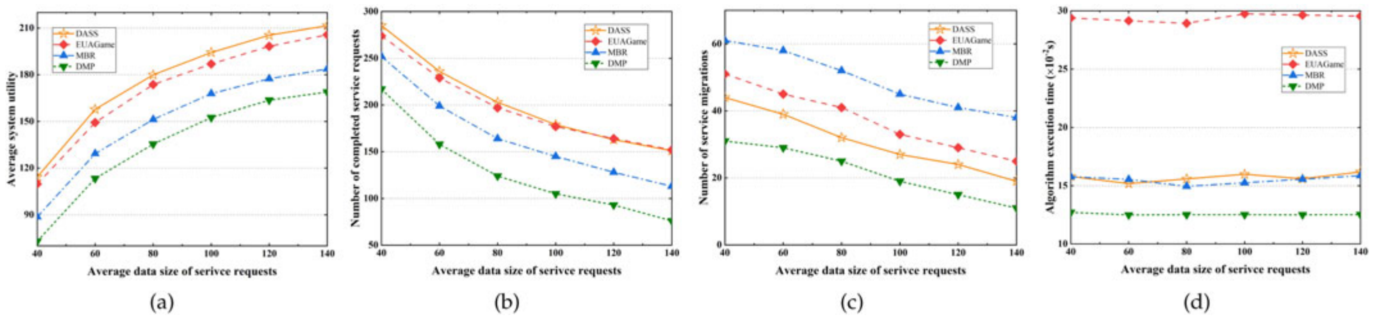


Fig. 7. Performance comparison with different data sizes of service requests under the real-world scenario: (a) Average system utility. (b) Number of completed service requests. (c) Number of service migrations. (d) Algorithm execution time.

relatively long time to finish each request, and the number of completed service requests reduces. The performance of DASS and EUAGame algorithms is similar. Compared with MBR and DMP methods, DASS algorithm can increase the number of completed service requests by 33 and 40 percent in the simulated scenario, respectively. While in the real-world scenario, the percentages are 25 and 62 percent, respectively. It demonstrates that our proposed DASS algorithm is able to provide pervasive offloading services in both user-intensive and user-sparse scenarios.

The Number of Service Migrations With Different Data Sizes of Service Requests. The influences of data size on the number of service migrations are illustrated in Figs. 6c and 7c. In general, it consumes more time slots to accomplish the task with a relatively large data size than that with a small data size. Correspondingly, the chance of migrations increases. However, the total number of completed requests reduces at the same time (see Figs. 6b and 7b). Therefore, the number of service migrations decreases slowly with the increasing data size of service requests. Considering two constraints of server storage capability and maximum tolerable latency, the conditions of large data size can be prevented from two aspects. First, tasks with large data size usually have long latency threshold. It is satisfactory that PEC servers accomplish tasks before deadline. Second, constrained by storage capability, merely limited tasks can be served by PEC servers.

Algorithm Execution Time With Different Data Sizes of Service Requests. It can be observed from Figs. 6d and 7d that the algorithm execution time is dependent of the request size, since the data size does not influence the time complexity of algorithms. In the simulated scenario, our proposed DASS algorithm can reduce the execution time by 39 percent compared with EUAGame approach, while increase that by 45 percent in the real-world scenario. Compared with the heuristic DMP method, the algorithm execution time increases by 16 and 10 percent in simulated and real-world scenarios, respectively. In summary, it consumes a relatively low latency for DASS algorithm to converge.

6.3 Result Summary

In this subsection, a brief overview of numerical results and final insights are presented to summarize the performance evaluation.

When service requests cannot be accomplished by one server and need to be migrated, it is necessary for PEC service providers to conduct prediction methods such as SAA algorithm to track the possible trajectories of MUs. In reality, the sampling space of SAA algorithm is limited for the regular urban road networks. Our proposed DASS can deal with both user-dense and user-sparse scenarios, and it performs well when the data size of tasks is relatively small (e.g., less than 60 MB).

Various characteristics can be taken into account by PEC service providers, including the system utility, service fraction and operating cost. Our proposed DASS algorithm outperforms other compared methods in terms of the system utility and service fraction, with a relatively low migration cost and algorithm execution time. In addition, the service fraction can be guaranteed in both user-intensive and user-sparse scenarios. With a reasonable partitioning of the edge network, it consumes relatively low latency for DASS algorithm to converge.

7 Conclusion

In this paper, we addressed the dynamic service placement problem in pervasive edge computing networks. Considering the high mobility of users and the limited storage capability of servers, we formulated the service placement issue as a long-term stochastic problem. To maximize the system utility, we proposed a dynamic storage-stable service placement algorithm. Theoretical analysis demonstrated that our proposed algorithm can be implemented in an online way, with a minor performance gap compared with the theoretically optimal solution. Performance evaluations were conducted based on both simulated and real-world scenarios. Numerical results showed that our proposed algorithm can achieve a satisfactory performance in terms of the obtained system utility, communication cost and algorithm execution time.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 61971084, 61931019, and 62001073, in part by Chongqing Talent Program CQYC2020058659, and in part by the National Natural Science Foundation of Chongqing under Grant cstc2019jcyjmsxmX0208.

References

[1] M. Patel *et al.*, "Mobile-edge computing introductory technical white paper," White Paper, Mobile-Edge Computing (MEC) Industry Initiative, pp. 1089–7801, 2014.
[2] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 411–425, Feb. 2021.
[3] Z. Ning *et al.*, "Partial computation offloading and adaptive task scheduling for 5G-enabled vehicular networks," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.3025116.
[4] X. Wang, Z. Ning, S. Guo, and L. Wang, "Multi-agent imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 411–425, 2021.
[5] Z. Ning *et al.*, "Mobile edge computing enabled 5G health monitoring for Internet of Medical Things: A decentralized game theoretic approach," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 463–478, 2021.
[6] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[7] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 762–772, Dec. 2013.
[8] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1440–1452, May 2016.
[9] M. Steiner *et al.*, "Network-aware service placement in a distributed cloud environment," in *Proc. ACM SIGCOMM Conf. Appl. Technologies Architectures Protocols Comput. Commun.*, 2012, pp. 73–74.
[10] 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.
[11] Q. Wu, X. Chen, Z. Zhou, and L. Chen, "Mobile social data learning for user-centric location prediction with application in mobile edge service migration," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7737–7747, Oct. 2019.

[12] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.

[13] J. Xu, X. Ma, A. Zhou, Q. Duan, and S. Wang, "Path selection for seamless service migration in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 9040–9049, Sep. 2020.

[14] 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.

[15] Y. Wu, Q. Yang, X. Liu, and K. S. Kwak, "Delay-constrained optimal transmission with proactive spectrum handoff in cognitive radio networks," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2767–2779, Jul. 2016.

[16] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 852–864, Apr. 2020.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, May 2016.

[18] N. Roy, A. Misra, S. K. Das, and C. Julien, "Determining quality-and energy-aware multiple contexts in pervasive computing environments," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 3026–3042, Oct. 2016.

[19] Z. Li and Y. Yang, "A novel network structure with power efficiency and high availability for data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 254–268, Feb. 2018.

[20] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.

[21] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[22] 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.

[23] J. Sahoo, M. A. Salahuddin, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on replica server placement algorithms for content delivery networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1002–1026, Second Quarter 2017.

[24] B. Yu and J. Pan, "A framework of hypergraph-based data placement among Geo-distributed datacenters," *IEEE Trans. Services Comput.*, vol. 13, no. 3, pp. 395–409, May/Jun. 2020.

[25] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 516–529, Jun. 2018.

[26] W. Chang and P. Wang, "Write-aware replica placement for cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 656–667, Mar. 2019.

[27] 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, Fourthquarter 2015.

[28] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[29] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1866–1880, Aug. 2019.

[30] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. Lau, "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.

[31] B. Dai *et al.*, "Multivariate Bernoulli distribution," *Bernoulli*, vol. 19, no. 4, pp. 1465–1483, 2013.

[32] G. Velkoski, M. Simjanoska, S. Ristov, and M. Gusev, "CPU utilization in a multitenant cloud," in *Proc. IEEE Eurocon*, 2013, pp. 242–249.

[33] H. Yin *et al.*, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1031–1045, Apr. 2017.

[34] S. Zhang, B. Di, L. Song, and Y. Li, "Sub-channel and power allocation for non-orthogonal multiple access relay networks with amplify-and-forward protocol," *IEEE Trans. Wireless Commun.*, vol. 16, no. 4, pp. 2249–2261, Apr. 2017.

[35] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, 2010.

[36] J. Ding, H. Liu, L. T. Yang, T. Yao, and W. Zuo, "Multiuser multivariate multiorder Markov-based multimodal user mobility pattern prediction," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4519–4531, May 2020.

[37] Z. Ning *et al.*, "Intelligent edge computing in Internet of Vehicles: A joint computation offloading and caching solution," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: 10.1109/TITS.2020.2997832.

[38] D. Son, A. Helmy, and B. Krishnamachari, "The effect of mobility-induced location errors on geographic routing in mobile ad hoc sensor networks: Analysis and improvement using mobility prediction," *IEEE Trans. Mobile Comput.*, vol. 3, no. 3, pp. 233–245, Jul./Aug. 2004.

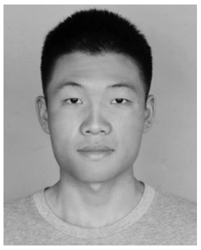
[39] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6301–6327, Oct, 2013.

[40] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.

[41] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[42] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.

Zhaolong Ning (Senior Member, IEEE) received the MS and PhD degrees from Northeastern University, Shenyang, China. He was an associate professor at the Dalian University of Technology, China. Currently, he is a distinguished professor with the Chongqing University of Posts and Telecommunications, China. He has published more than 100 scientific papers in international journals and conferences. His research interests include Internet of vehicles, mobile edge computing, and artificial intelligence.



Peiran Dong received BS degree from the Dalian University of Technology, Dalian, China, in 2018. He is currently working toward the MS degree in software engineering at the Dalian University of Technology. His research interests include mobile edge computing, network computation offloading, and resource management.

Xiaojie Wang received the MS degree from Northeastern University, China, in 2011, and the PhD degree from the Dalian University of Technology, Dalian, China, in 2019. From 2011 to 2015, she was a software engineer with NeuSoft Corporation, China. Currently, she is a distinguished professor at the Chongqing University of Posts and Telecommunications, China. Her research interests are wireless networks, mobile edge computing, and machine learning.

Shupeng Wang received the MS and PhD degrees from the Harbin Institute of Technology, Harbin, China, in 2004 and 2007, respectively. He is currently a senior engineer with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing, China. His research interests include big data management and analytics, network storage, etc.

Xiping Hu received the PhD degree in electrical and computer engineering from the University of British Colombia, Vancouver, Canada. He is a professor with Lanzhou University. He was the cofounder and CTO of Bravolol Limited, Hong Kong, a leading language learning mobile application company with more than 100 million users, and listed as top two language education platform globally. He has around 90 papers published and presented in prestigious conferences and journals. His research areas consist of distributed intelligent systems, crowdsensing, social networks, and cloud computing.

Song Guo (Fellow, IEEE) received the PhD degree in computer science from the University of Ottawa. He was a professor at the University of Aizu. He is a full professor with the Department of Computing, the Hong Kong Polytechnic University. His research interests include big data, cloud computing and networking, and distributed systems with more than 400 papers published in major conferences and journals. His work was recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and other five best paper awards from IEEE/ACM conferences. He was an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and an IEEE ComSoc distinguished lecturer. He is currently on the editorial board of the *IEEE Transactions on Emerging Topics in Computing*, *IEEE Transactions on Sustainable Computing*, *IEEE Transactions on Green Communications and Networking*, and *IEEE Communications*. He also served as general, TPC and symposium chair for numerous IEEE conferences. He currently serves as an officer for several IEEE ComSoc Technical Committees and a director in the ComSoc Board of Governors.

Tie Qiu (Senior Member, IEEE) received the PhD degree in computer science from the Dalian University of Technology, in 2012. He is currently a full professor with the School of Computer Science and Technology, Tianjin University, China. He was a visiting professor at electrical and computer engineering at Iowa State University in US (2014–2015). He serves as an associate editor of *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, area editor of *Ad Hoc Networks* (Elsevier), associate editor of *IEEE Access Journal*, *Computers and Electrical Engineering* (Elsevier), a guest editor of *Future Generation Computer Systems*. He has authored/co-authored nine books, more than 150 scientific papers in international journals, and conference proceedings.

Bin Hu (Senior Member, IEEE) is currently a professor with Lanzhou University and a guest professor with ETH Zurich, Switzerland. He is an IET fellow, co-chairs of IEEE SMC TC on Cognitive Computing, and member with the Large of ACM China, vice president of International Society for Social Neuroscience (China committee) etc. He has published more than 100 papers in peer reviewed journals, conferences, and book chapters including *Science*, *Journal of Alzheimer's Disease*, *PLoS Computational Biology*, *IEEE Trans.*, *IEEE Intelligent Systems*, AAAI, etc. He has served as chairs/co-chairs in many IEEE international conferences/workshops, and associate editors in peer reviewed journals on Cognitive Science and Pervasive Computing, such as *IEEE Trans. Affective Computing*, *Brain Informatics*, *IET Communications*, etc.

Ricky Y. K. Kwok (Fellow, IEEE) received the BSc degree in computer engineering from the University of Hong Kong, in 1991, and the MPhil and PhD degrees, computer science, from the Hong Kong University of Science and Technology (HKUST), in 1994 and 1997, respectively. His research focus has been on designing efficient communication protocols and robust resources management algorithms toward enabling large scale distributed mobile computing. In these research areas, he has authored one textbook, co-authored another two textbooks, and published more than 200 technical papers in various leading journals, research books, and refereed international conference proceedings. He is a fellow of the HKIE and the IET. From March 2006 to December 2011, He served on the Editorial Board of the *Journal of Parallel and Distributed Computing* as a subject area editor in *Peer-to-Peer Computing*. He also served as an associate editor for the *IEEE Transactions on Parallel and Distributed Systems* from January 2013 to December 2016.