CHINA COMMUNICATIONS

Lyapunov-guided Optimal Service Placement in Vehicular Edge Computing

Chaogang Tang[1], Yubin Zhao[2], Huaming Wu[3*]

[1] School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China
[2] The Center for Applied Mathematics, Tianjin University, Tianjin 300072, China
[3] School of Microelectronics Science and Technology, Sun Yat-Sen University, Zhuhai 519082, China
[*] The corresponding author, email: whming@tju.edu.cn

Abstract: Vehicular Edge Computing (VEC) brings the computational resources in close proximity to the service requestors and thus supports explosive computing demands from smart vehicles. However, the limited computing capability of VEC cannot simultaneously respond to large amounts of offloading requests, thus restricting the performance of VEC system. Besides, a mass of traffic data can incur tremendous pressure on the front-haul links between vehicles and the edge server. To strengthen the performance of VEC, in this paper we propose to place services beforehand at the edge server, e.g., by deploying the services/tasks-oriented data (e.g., related libraries and databases) in advance at the network edge, instead of downloading them from the remote data center or offloading them from vehicles during the runtime. In this paper, we formulate the service placement problem in VEC to minimize the average response latency for all requested services along the slotted timeline. Specifically, the time slot spanned optimization problem is converted into per-slot optimization problems based on the Lyapunov optimization. Then a greedy heuristic is introduced to the drift-plus-penalty based algorithm for seeking the approximate solution. The simulation results reveal its advantages over others in terms of optimal values and our strategy can satisfy the long-term energy constraint.

Keywords: Vehicular edge computing; Service placement; Response latency; Computational resources

Received: ***
Revised: ***
Editor: ***

I. INTRODUCTION

Various advanced Information and Communication Technologies (ICT) are applied and integrated into Intelligent Transportation Systems (ITS) to pursue harmonious unification of humans, vehicles and roads. Many benefits can be obtained from ITS, including guaranteeing traffic safety, raising transportation efficiency, improving the transportation environment and driving experience. In this context, a wide variety of data can be gathered by mounted facilities in smart vehicles (e.g., SIM cards, GPS, sensors, and cameras). Furthermore, the consequent vehicular tasks have sprung up, which require computational resources and services for successful execution. Smart vehicles are usually equipped with computing facilities such as On-Board Unit (OBU) to satisfy the computational demands of these tasks. However, such "computers on wheels" can only offer limited computing capabilities, due to restricted physical size and energy supply.

To alleviate the contradiction between the skyrocketing demands for computational resources and the limited computing capabilities of vehicles themselves, it becomes increasingly dominant that computational resources and services are provisioned outside the vehicles. For example, Vehicular Cloud Computing (VCC) and Vehicular Edge Computing (VEC) are two newly emergent computing paradigms in recent years, which are intended for addressing the above contra-

diction. The difference between VCC and VEC lies in that VEC shifts the computational capability from the remote data center to the network edge and provides computing services in close proximity to the resource requestors (i.e., vehicles). Comparatively speaking, VEC can better cater to the delay-strict requirements of vehicles tasks. In particular, vehicles offload their tasks/applications to the edge server for execution in VEC and the edge server is usually deployed at the Road-Side Unit (RSU) in a densely populated area. However, due to explosive growth in vehicular tasks and applications, tremendous pressure has been imposed on the computing capabilities of the edge server as well as the front-haul links between vehicles and an accessible computational access point, e.g., RSU. This is because the amount of computational resources in VEC is not unlimited compared to that at the cloud center.

Accordingly, we propose to place services in advance at the edge server. Service placement, also termed service deployment, strives to deploy the services/tasks-oriented data (e.g., related libraries and databases) in advance at the network edge (e.g., RSUs), instead of downloading them from remote data center during the runtime or rush hours. When a vehicle sends its offloading request to the edge server, the execution result can be directly retrieved, if the corresponding service has been placed at the edge server. Otherwise, the corresponding service may be offloaded from the vehicle or downloaded from the data center, which increases both the transmission and calculation delay. As a result, the response delay is much longer, compared to the case when the service is cached.

Although there is an extensive literature on task offloading and service placement in VEC [1–4], they seldom consider factors which may increase the difficulty of service placement. Such factors usually include limited sojourn time for vehicles, limited wireless coverage and vehicular offloading requests featured by temporal and spatial variation. In addition, most of the current works on service caching assume that service caching does not incur energy consumption at the edge server, which, however, does not always hold in the task-oriented caching scenarios [5, 6]. As a contrast, in this paper we not only take into account the above factors but also the evaluation of service placement in the long run.

In this paper, we strive to make service placement decision with a time effect and strategic update in a scenario where vehicles are covered by multiple RSUs owing to an increasing density of RSUs. Specifically, services are placed at RSUs along the slotted timeline such that the average response latency for all the requested services can be optimized. The major contributions are summarized as follows.

- A generic approach is put forward for improving the performance of VEC in terms of response latency, with the help of strategic service placement. By formulating the mathematical model, our goal is to minimize the average response latency of requested services along the slotted timeline in this paper.
- Multiple constraints are considered in the optimization problem, especially the overall constraint of energy consumption along the infinite time-slotted horizon. The Lyapunov optimization-based technology is leveraged for converting the long-term energy constraint into per-slot ones. Based on this, an online algorithm is designed to search for the approximate optimal solution in this paper.
- Extensive simulation is conducted for investigating and evaluating the performance of the proposed service placement strategy. Simulation results have revealed that our strategy can achieve better performance compared to other strategies.

The rest of the paper is organized as follows. Some related works are reviewed in Section II. The system model is mathematically formulated in Section III, and the optimization problem is also given in Section IV. An online approach for service placement is put forward in Section V, followed by the simulation evaluation in Section VI. Finally, the conclusion is drawn in Section VII.

II. RELATED WORK

Due to restricted physical size and energy supply of smart devices (e.g., OBU), there are increasingly strong demands for shifting the task execution from the local to the third party with abundant computational resources. Such a tendency has given birth to several new computing paradigms represented by edge computing. Extensive attention has been paid to this

research field, aiming to optimize the performance of task offloading and service outsourcing from different angles [7]. In the meanwhile, service placement usually plays an important role in improving the performance of the systems when tasks are offloaded and resources are scheduled in the cloud-edge networks [8–12].

Mobile Edge Computing (MEC), derived from edge computing, brings the computational resources close to the devices, and enables tasks to be accomplished with shorter response latency compared to cloud computing. To further enhance the performance of MEC, Li *et al.* [1] jointly optimized task offloading, service placement and resources deployment at the edge in MEC. To improve the quality of service such as response latency, service providers usually deploy services at the network edges. In this context, Panadero *et al.* [13] studied the service placement at the edge server by minimizing the total cost for task offloading which includes the costs on service placement, the throughput of edge server, and the energy consumption. In the meanwhile, the strict requirement for user-perceived delay should be satisfied. They proposed a heuristic algorithm to solve the formulated Mixed Integer Linear Programming problem.

Similarly, the Internet of Things (IoT) devices have increasing demands for services characterized by ultra-low response latency. In this context, minimization of response latency while meeting multiple constraints has become the main focus in the existing literature. However, current works seldom consider the load-balancing issue which has played a key role in cost-efficient system management. Therefore, an optimization problem with two optimization objectives was proposed in [14], aiming to place IoT services with load balance for improving the quality of service that is defined as a function of deadline violation, service deployment, and unhosted services.

Task execution at the network edges brings benefits to the task requestors, but it also causes serious pressure on the limited computational resources of the edge servers. How to efficiently schedule these resources for the requestors has become one of the prime challenges. Different from the previous works which try to jointly optimize the service placement and request routing, Yuan *et al.* [15] have considered the non-negligible operating expenses caused by service placement. Specifically, they introduce a two timescale framework to minimize these costs during frequent cross-cloud service replacement and replication, and further design a greedy algorithm to address the service placement problem. Community networks have attracted lots of attention recently. These networks are usually constructed and managed voluntarily at the edge, and featured by irregular topology and resource heterogeneity. In view of this, Panadero *et al.* [16] proposed a service placement strategy, which can place services in decentralized community networks.

Complicated services can usually be divided into independent components which can be separately accomplished at different edge servers in a cooperative fashion. Previous works usually assumed services are unsplittable or ignored the fact that edge servers can be geographically isolated. Furthermore, the interference caused by services sharing the same physical nodes/links was also ignored, which however can cause long computation delay. Accordingly, Han *et al.* [17] paid attention to the Interference-Aware (IA) service placement while assuming that services can be composed of multiple components in the edge-cloud networks.

Apart from the computational resources, storage and networking resources are both required for data-intensive applications such as machine learning tasks. Furthermore, the service (libraries, databases and codes) placement strategy should be updated over time. Accordingly, a two-time-scale framework is put forward in [18], aiming to jointly optimize service placement and request allocating. Moreover, a polynomial-time service placement algorithm is designed in real-time response to the service requests. Simulation results reveal that their approach achieves 90% of the optimal performance. To realize service migration in dynamic mobile networks, pervasive edge computing is attracting more and more attention recently. Against this background, Ning *et al.* [19] aimed to maximize the utility value of the edge system. Specifically, they consider both storage capacity and response latency in the optimization problem. Lyapunov optimization is introduced for converting it into instant subproblems. Moreover, two approximation algorithms are used to obtain the average system utility in the future and make decisions on service placement, respectively.

Due to the increasingly unoccupied resources on

UAVs, more and more UAVs can act as the edge servers to process the offloaded tasks by the onboard processors [20–22]. Thus, services should be placed beforehand. He *et al.* [20] aimed to jointly optimize the service placement and resource allocation problem in UAV-assisted edge computing, which was modeled as a mixed-integer nonlinear programming problem and divided into two subproblems which are proven to be submodular and convex, respectively. Thus, a general alternative optimization is adopted to solve this problem.

III. SYSTEM MODEL

A considered scenario is depicted in Fig. 1. Multiple RSUs are scattered in the densely populated area. In addition, the performance of them is enhanced by deploying edge servers that are rich in computational resources, as denoted in the figure. Thus, services can be provided at the edge of the network. Considering the mobility of vehicles, the strategy for service placement, i.e., which services should be deployed at which RSUs, should be updated periodically. Therefore, we divide the timeline $\mathcal{T}$ into T time slots in this paper, and each time slot lasts τ seconds, i.e., $\mathcal{T} = \{0, \cdots, T-1\}$. The service placement can be updated within each time slot, so as to reflect its dynamics. We denote the set of RSUs by $\mathcal{N} = \{1, \cdots, N\}$, where N is the number of RSUs in the considered model. In addition, we further assume that the area considered in this paper can be divided into M disjoint sub-areas (regions), indexed by $\mathcal{M}$. Due to the increasing density of RSU deployment, it is likely that one sub-area or region can be covered by multiple RSUs.

Let $\mathcal{R}_m (\subset \mathcal{N})$ denote the set of RSUs that can cover the region m. Thus, vehicles with offloading requests at region m can resort to RSU $n \in \mathcal{R}_m$. Assume that there are K computing services that can be deployed at RSUs, indexed by $\mathcal{K} = \{1, \cdots, K\}$. Each service k can be represented by a 2-tuple of (d_k, s_k), where d_k and s_k denote the task-input data size and averagely required workload (e.g., the CPU cycles), respectively. Each RSU n can be represented by a 2-tuple of (D_n, f_n), where D_n and f_n denote the caching and processing capabilities of RSU n, respectively.

Definition 1. ***Service Placement Decision***. *Service placement in this paper refers to the deployment of services at the edge server by downloading service-oriented data such as parameters, libraries and databases in advance from the remote data center. Let $\delta_{k,n}^t$ be a binary variable to indicate whether the service k is deployed at RSU n in time slot t. $\delta_{k,n}^t = 1$, if service k is deployed at RSU n in time slot t; and 0, otherwise. Define $\boldsymbol{\delta}_n^t = \{\delta_{1,n}^t, \cdots, \delta_{K,n}^t\}$ as the service placement decision of RSU n for all services in $\mathcal{K}$ in time slot t, and $\boldsymbol{\delta}^t = \{\boldsymbol{\delta}_1^t, \cdots, \boldsymbol{\delta}_N^t\}$ as the service placement decisions of all RSUs in $\mathcal{N}$ in time slot t.*

Note that it is impractical for RSU n to deploy all the services simultaneously, since the storage (caching) capability (i.e., D_n) is limited. In particular, the service placement decisions should meet the following constraint:

$$\sum_{k=1}^{K} \delta_{k,n}^t d_k \leq D_n, \quad \forall t \in \mathcal{T}, \quad \forall n \in \mathcal{N}. \tag{1}$$

The above equation indicates that the total amount of task-input data of all the services placed at RSU n should not exceed its caching capability D_n in each time slot. We investigate the performance of service placement in vehicle edge computing in regions instead of individual vehicles [2], since the statistical information on vehicular offloading requests can be better captured and evaluated in regions than individual vehicles, and this way can better cater to service placement featured by the long-term process. As a result, it is no longer adequate to assume that the number of vehicular offloading requests is constant as most of existing literature did [3, 5]. In this paper, we assume that the vehicular task arrivals, i.e., the arrivals of the offloading requests generated by vehicles in the corresponding regions, follow a Poisson process. In particular, let $\lambda_{k,m}^t$ denote the arrival rate of vehicular tasks requesting service k in region m in time slot t.

Let $\mathcal{R}_{k,m}^t \subseteq \mathcal{R}_m$ denote the set of RSUs that have deployed service k in region m in time slot t. Thus, vehicular offloading requests in region m can resort to the RSU in $\mathcal{R}_{k,m}^t$. Thus, two cases may occur as follows. One is that the set $\mathcal{R}_{k,m}^t$ is empty. Then, we assume that the vehicular offloading requests will be forwarded to the cloud center where the tasks are served by corresponding services. The other case is that, the set $\mathcal{R}_{k,m}^t$ is not empty, i.e., there are multiple RSUs that can serve the vehicular offloading requests. For simplicity, the vehicular offloading requests are as-

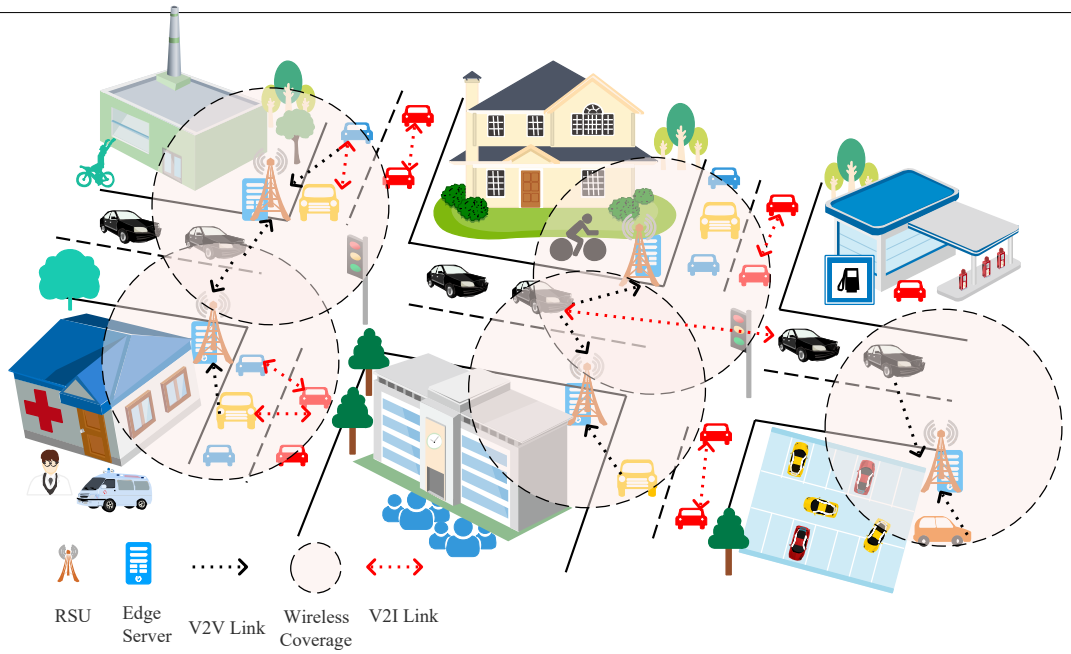

Figure 1. *A VEC Application scenario considered in this paper*

sumed to be evenly distributed among RSUs in $\mathcal{R}^t_{k,m}$ [2]. Therefore, we define $w^t_{k,n}$ as the average workloads required by service k to RSU n in time slot t, given as:

$$w^t_{k,n} = \begin{cases} 0 & \bigcup_{m=1}^{M} \mathcal{R}^t_{k,m} = \emptyset, \\ \sum_{m=1}^{M} I_{n,m} \frac{s_k \lambda^t_{k,m}}{|\mathcal{R}^t_{k,m}|} & \bigcup_{m=1}^{M} \mathcal{R}^t_{k,m} \neq \emptyset. \end{cases} \tag{2}$$

where $\bigcup_{m=1}^{M} \mathcal{R}^t_{k,m} = \emptyset$ means that there are no regions $m \in \mathcal{M}$ that have placed service k. $I_{n,m}$ is an indicator to denote whether the RSU n covers the region m. It can be defined as:

$$I_{n,m} = \begin{cases} 1 & n \in \mathcal{R}_m, \\ 0 & n \notin \mathcal{R}_m. \end{cases} \tag{3}$$

Combining the service placement decision $\delta^t_{k,n}$, $w^t_{k,n}$ can be further rewritten as:

$$w^t_{k,n} = \delta^t_{k,n} \sum_{m=1}^{M} I_{n,m} \frac{s_k \lambda^t_{k,m}}{|\mathcal{R}^t_{k,m}|}, \quad \forall k \in \mathcal{K}, \forall n \in \mathcal{N}. \tag{4}$$

Service placement at the edge server in advance aims to reduce the response delay and further improve the quality of service (QoS) and the quality of experience (QoE). However, it shall be noted that, the service deployment and placement at the edge, e.g., by downloading task-input data, libraries and databases pertaining to the corresponding services from the data center, will also incur costs such as energy consumption at the edge. We will separately discuss the response delay and energy consumption models in what follows.

3.1 Response Delay Model

The average response delay for the tasks processed at the edge mainly consists of the queueing delay and computation delay, when the corresponding services have already been placed at the edge server in advance. We have considered the queueing delay in the paper, owing to the fact that the computational resources at RSUs cannot process all the tasks in parallel, especially when the number of vehicular offloading requests is huge. Furthermore, the queueing delay is one of the causes that incur long response latency.

Since the vehicular task arrival of each service type follows a Poisson process with rate $\lambda^t_{k,m}$ and the offloading requests are evenly distributed among RSUs in $\mathcal{R}^t_{k,m}$, the overall task arrival at RSU n for service k is also a Poisson process, with the rate $\lambda^t_{k,n}$ given as:

$$\lambda^t_{k,n} = \sum_{m=1}^{M} I_{n,m} \frac{\lambda^t_{k,m}}{|\mathcal{R}^t_{k,m}|}, \quad \forall k \in \mathcal{K}, \forall n \in \mathcal{N}. \tag{5}$$

The service time of one task of service type k at RSU n can be expressed as s_k/f_n. Accordingly, the service rate $\mu_{k,n}$ is:

$$\mu_{k,n} = \frac{f_n}{s_k}, \quad \forall k \in \mathcal{K}, \forall n \in \mathcal{N}, \tag{6}$$

where f_n denotes the processing frequency of RSU n. Specifically, we evaluate and analyze the response delay by modeling this process as an M/M/1 queue. Thus the response delay, i.e., the average sojourn time (waiting time and service time) for service k at RSU n in time slot t, is given as:

$$rd_{k,n}^t = \frac{s_k}{f_n - s_k \sum_{m=1}^{M} \frac{I_{n,m}\lambda_{k,m}^t}{|\mathcal{R}_{k,m}^t|}}. \tag{7}$$

The above equation implies that service k has been placed at RSU n. On the other hand, it is also possible that service k is not placed at RSU n. In this case, the offloading request will be served at the cloud center. Due to the sufficient computing resources and powerful computing capabilities at the cloud center, we thus assume that the calculation delay at the cloud center can be ignored. The response delay, denoted by $rd_{k,c}^t$, is mainly caused by the transmission delay when data and execution results are transmitted over the backbone network using the backhaul links. Such transmission delay can be estimated and predicted based on historical statistical data and highly efficient machine learning approaches such as auto-regression analysis. Hence, the average response delay for service k in time slot t rd_k^t is:

$$rd_k^t = \frac{1}{N}\sum_{n=1}^{N} \delta_{k,n}^t rd_{k,n}^t + (1 - \delta_{k,n}^t) rd_{k,c}^t. \tag{8}$$

3.2 Energy Consumption Model

As mentioned earlier, the service placement at the edge in advance incurs negligible energy consumption, including the static energy consumption and dynamic energy consumption. The static energy, denoted by α_n, is consumed for maintaining virtual resources allocated to service k, which is independent of the workload of the service. The dynamic energy $\beta_{k,n}^t$ is consumed for performing the service at run time. Thus, the total energy consumption for service k at RSU n in time slot t (i.e., the static energy consumption plus the dynamic energy consumption) can be expressed as:

$$\begin{aligned} e_{k,n}^t &= \alpha_n + \varsigma\eta w_{k,n}^t (f_n)^2 \\ &= \alpha_n + \varsigma\eta\delta_{k,n}^t (f_n)^2 \sum_{m=1}^{M} I_{n,m} \frac{s_k \lambda_{k,m}^t}{|\mathcal{R}_{k,m}^t|}, \end{aligned} \tag{9}$$

where the second part on the right-hand side represents the dynamic energy consumption $\beta_{k,n}^t$, ς represents the effectively switched capacitance coefficient, and η is the number of cycles needed to perform one task-input bit from n. The energy consumption for computation undertaking at the edge server is supposed to be lower than the given threshold, so the residual energy can be kept for other purposes such as vehicle-to-infrastructure and pedestrian-to-infrastructure communications. Thus, the following constraint should be satisfied:

$$\sum_{k=1}^{K} e_{k,n}^t \le e_{max,n}^t, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \tag{10}$$

where $e_{max,n}^t$ is the maximal energy consumption that RSU n can tolerate to perform the tasks. Therefore, for arbitrary time slot t, the total energy consumption for undertaking all the services at RSU n should not exceed this energy threshold. For easy discussion, we assume that the energy threshold is the same for each time slot at RSU n.

IV. PROBLEM FORMULATION

Our goal in this paper is to minimize the overall response delay of services along the infinite time slots, by adjusting the strategy of service placement at RSUs within each time slot. In particular, our optimization

problem can be formulated as:

$$(\mathcal{P}1) \quad \min_{\boldsymbol{\delta}^t, \forall t} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k=1}^{K} r d_k^t$$

$$\text{s.t.}: \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k=1}^{K} e_{k,n}^t \leq \mathcal{E}_n, \quad \forall n \in \mathcal{N}, \quad (11)$$

$$f_{n,min} \leq f_n \leq f_{n,max}, \quad \forall n \in \mathcal{N}, \quad (12)$$

$$I_{n,m} \in \{0,1\} \quad \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \quad (13)$$

$$\delta_{k,n}^t \in \{0,1\} \quad \forall k \in \mathcal{K}, \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \quad (14)$$

$$\text{Constraints} \quad (1), \ (10), \quad (15)$$

where the inequation (11) represents that the long-term energy consumption at RSU n should not exceed the threshold $\mathcal{E}_n$, for the reason that the evaluation of service placement strategy needs a long process as mentioned earlier. The computational resources allocated for service deployment and execution should be bounded as shown in the inequation (12), owing to the limited computing capabilities at the edge server in contrast to the cloud center. The number of services that are placed is restricted by the limited storage capacity of the edge server. In addition, per-slot energy consumption for RSU n is also bounded by $e_{max,n}^t$ as shown in Eq. (10).

The difficulties in addressing the problem $\mathcal{P}1$ are summarized as follows. First, the entire information, such as the offloading requests in all time slots, is required for optimally solving this problem. However, it is difficult to obtain this information of the future time slots in the current time slot. In addition, $\mathcal{P}1$ is essentially an integer linear programming that is computationally prohibitive even if the entire information is acquired a priori. Last but not least, it is quite difficult to handle the constraint (11), since it similarly requires the entire information in all the time slots. Accordingly, an online approach is needed for addressing this problem, such that the response delay can be optimized by placing services at the edge within each time slot.

V. ONLINE ALGORITHM FOR SERVICE PLACEMENT

To address the above challenges, we in this section present an online algorithm, named Online service Placement for vehicular Edge Computing (OPEC). Specifically, the Lyapunov optimization technology is introduced for converting the infinite time-slotted (global) energy consumption constraint (11) into respective constraints within each time slot. Thus, the optimization for $\mathcal{P}1$ can be solved approximately within each time slot.

5.1 Lyapunov-based Online Algorithm

To covert the global constraint into local ones, n energy migration queues $Q_n(n \in \mathcal{N})$ are constructed, which can be recursively defined as:

$$Q_n(t+1) = \max[Q_n(t) - \mathcal{E}_n, 0] + \sum_{k=1}^{K} e_{k,n}^t, \forall n \in \mathcal{N}, \quad (16)$$

where $Q_n(t)$ is a numerical value to denote the queue backlog of RSU n in time slot t. Actually, $Q_n(t)$ can indicate how far the energy consumption in current time slot deviates from the global energy constraint $\mathcal{E}_n$, e.g., a larger value of $Q_n(t)$ for RSU n always denotes a sharper deviation from $\mathcal{E}_n$. Note that we assume that $Q_n(0) = 0$ holds, $\forall n \in \mathcal{N}$.

Lemma 1. *For an arbitrary RSU $n \in \mathcal{N}$, given the corresponding queue backlog $Q_n(t)$ in time slot t, the following inequality always holds*

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n] \leq \frac{1}{T} \mathbb{E}[Q_n(T)].$$

Proof. : The lemma can be proven based on the following two cases.

- Case 1: $Q_n(t) \geq \mathcal{E}_n$. In this case, we have $Q_n(t+1) = \max[Q_n(t) - \mathcal{E}_n, 0] + \sum_{k=1}^{K} e_{k,n}^t = Q_n(t) - \mathcal{E}_n + \sum_{k=1}^{K} e_{k,n}^t$, so we can obtain $\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n = Q_n(t+1) - Q_n(t)$.
- Case 2: $Q_n(t) < \mathcal{E}_n$. In this case, $Q_n(t+1) = \max[Q_n(t) - \mathcal{E}_n, 0] + \sum_{k=1}^{K} e_{k,n}^t = \sum_{k=1}^{K} e_{k,n}^t$ and $\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n = Q_n(t+1) - \mathcal{E}_n < Q_n(t+1) - Q_n(t)$. As a result, $\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n \leq Q_n(t+1) - Q_n(t)$ holds, $\forall t \in \mathcal{T}$. Take the expectation of this inequality and then summarize it over $\forall t \in \mathcal{T}$, we

have:

$$\sum_{t=0}^{T-1}\mathbb{E}\left[\sum_{k=1}^{K}e_{k,n}^{t}-\mathcal{E}_n\right]$$
$$\leq\sum_{t=0}^{T-1}\mathbb{E}[Q_n(t+1)-Q_n(t)]=\mathbb{E}[Q_n(T)].$$

Thus, $\forall n\in\mathcal{N}$, we can obtain

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\sum_{k=1}^{K}e_{k,n}^{t}-\mathcal{E}_n\right]\leq\frac{1}{T}\mathbb{E}[Q_n(T)].$$

Define the vector $\boldsymbol{Q}(t)=\{Q_1(t),\cdots,Q_N(t)\}$ as the queue backlogs of N RSUs in time slot t. Then, the Lyapunov function with regards to (w.r.t.) $\boldsymbol{Q}(t)$ is defined as:

$$\mathcal{F}(\boldsymbol{Q}(t))=\frac{1}{2}\sum_{n=1}^{N}Q_n^2(t). \tag{17}$$

The Lyapunov function is actually the function of the queue backlogs of all the RSU $\mathcal{N}$ in time slot t. We can further define *Lyapunov drift* as the increment of the queue backlogs in two consecutive time slots, given as:

$$\triangle(\boldsymbol{Q}(t))\triangleq\mathbb{E}[\mathcal{F}(\boldsymbol{Q}(t+1))-\mathcal{F}(\boldsymbol{Q}(t))|\boldsymbol{Q}(t)]. \tag{18}$$

The Lyapunov drift can actually indicate the fluctuation of two consecutive states. From the perspective of system stability, the smaller the Lyapunov drift, the better the system stability. However, to minimize the Lyapunov drift faces the similar challenge, since it also needs the information in the future time slots. For example, the optimization of $\triangle(\boldsymbol{Q}(t))$ needs the future information $\mathcal{F}(\boldsymbol{Q}(t+1))$. To avoid involving the future information, we can actually relax it by determining its upper bound. In the next, we will show how to find the upper bound of the Lyapunov drift according to the following lemma [23].

Lemma 2. *Given four non-negative real numbers A, B, C and m, they satisfy $C=\max\{B-m,0\}+A$, then $C^2\leq A^2+B^2+m^2-2B(m-A)$.*

Based on this Lemma, we have

$$\begin{aligned}
&\triangle(\boldsymbol{Q}(t))=\mathbb{E}[\mathcal{F}(\boldsymbol{Q}(t+1))-\mathcal{F}(\boldsymbol{Q}(t))|\boldsymbol{Q}(t)]\\
&=\frac{1}{2}\mathbb{E}[\sum_{n=1}^{N}[\max[Q_n(t)-\mathcal{E}_n,0]+\sum_{k=1}^{K}e_{k,n}^{t}]^2\\
&-\sum_{n=1}^{N}Q_n^2(t)|\boldsymbol{Q}(t)]\\
&\leq\frac{1}{2}\mathbb{E}[\sum_{n=1}^{N}[\mathcal{E}_n^2+(\sum_{k=1}^{K}e_{k,n}^{t})^2\\
&+2Q_n(t)(\sum_{k=1}^{K}e_{k,n}^{t}-\mathcal{E}_n)]|\boldsymbol{Q}(t)]\\
&=D-\sum_{n=1}^{N}\mathcal{E}_nQ_n(t)+\mathbb{E}[\sum_{n=1}^{N}Q_n(t)\sum_{k=1}^{K}e_{k,n}^{t}|\boldsymbol{Q}(t)]
\end{aligned} \tag{19}$$

where

$$\begin{aligned}
D&=\frac{1}{2}\sum_{n=1}^{N}\mathcal{E}_n^2+\frac{1}{2}\mathbb{E}\left[\sum_{n=1}^{N}\left(\sum_{k=1}^{K}e_{k,n}^{t}\right)^2|\boldsymbol{Q}(t)\right]\\
&\leq\frac{1}{2}\sum_{n=1}^{N}\mathcal{E}_n^2+\frac{1}{2}\mathbb{E}\left[\sum_{n=1}^{N}(e_{max,n}^{t})^2|\boldsymbol{Q}(t)\right]\\
&=\frac{1}{2}\sum_{n=1}^{N}\left[\mathcal{E}_n^2+(e_{max,n}^{t})^2\right]\triangleq B.
\end{aligned} \tag{20}$$

.

Therefore, the upper bound of the Lyapunov drift can be given as:

$$\begin{aligned}
\triangle(\boldsymbol{Q}(t))\leq B&-\sum_{n=1}^{N}\mathcal{E}_nQ_n(t)\\
&+\mathbb{E}\left[\sum_{n=1}^{N}Q_n(t)\sum_{k=1}^{K}e_{k,n}^{t}|\boldsymbol{Q}(t)\right].
\end{aligned} \tag{21}$$

It is noticeable that the upper bound of $\triangle(\boldsymbol{Q}(t))$ no longer depends upon the information in future time slots such as $\boldsymbol{Q}(t+1)$. Then, we introduce the *drift-plus-penalty* term which integrates the Lyapunov drift into the optimization objective in per time slot, given as below:

$$\Theta(t)=\triangle(\boldsymbol{Q}(t))+V\mathbb{E}\left[\sum_{k=1}^{K}rd_k^t|\boldsymbol{Q}(t)\right], \tag{22}$$

where V is a positive numeric value to denote the preference between energy consumption and response delay. Combining the inequality (21), we have:

$$\Theta(t) \leq B - \sum_{n=1}^{N} \mathcal{E}_n Q_n(t) + \mathbb{E}\left[\sum_{n=1}^{N} Q_n(t) \sum_{k=1}^{K} e_{k,n}^t + V \sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)\right]. \tag{23}$$

We now turn our attention to the optimization of the supremum of the drift-plus-penalty term, i.e., the right hand side of the inequality (23). Since B, $\mathcal{E}_n$ and $Q_n(t)$ are independent of the service placement decisions and thus can be obtained in time slot t, to optimize the supremum of the drift-plus-penalty is equivalent to the optimization of the following problem:

$$(\mathcal{P}2) \quad \min_{\forall t, \boldsymbol{\delta}^t} \left\{ \sum_{n=1}^{N} Q_n(t) \sum_{k=1}^{K} e_{k,n}^t + V \sum_{k=1}^{K} r d_k^t \right\}$$
$$\text{s.t.}: \quad (12), (13), (14), (15) \tag{24}$$

Therefore, we can solve the problem $\mathcal{P}1$ approximatively by solving the problem $\mathcal{P}2$. Compared to $\mathcal{P}1$, it is relatively easier to solve $\mathcal{P}2$, because the time slot spanned energy constraint has been converted into per-slot ones. An important issue is naturally posed, i.e., what is the degree of approximation of the solution of $\mathcal{P}2$ towards that of $\mathcal{P}1$.

Theorem 1. *If there exists a solution to the problem $\mathcal{P}1$, the service placement profile obtained by solving $\mathcal{P}2$ over time slots $t \in \{0, \cdots, T-1\}$ can also approximately optimize the problem $\mathcal{P}1$ and the optimality gap which denotes the difference between the two solutions is bounded by $O(1/V)$.*

Proof. : Since there is a solution to $\mathcal{P}1$, there exits a service placement profile $\boldsymbol{\delta}^{t*}$ in time slot t, and the corresponding global optimum l^*, which respectively satisfy 1) $\sum_{k=1}^{K} e_{k,n}^t \leq \mathcal{E}_n$, $\forall n \in \mathcal{N}$ and 2) $l^* = \min_{\boldsymbol{\delta}^t, \forall t} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} r d_k^t$, based on [24]. Accordingly, we have:

$$\begin{aligned}
\Theta(t) &= \triangle(\boldsymbol{Q}(t)) + V\mathbb{E}\left[\sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)\right] \\
&\leq B - \sum_{n=1}^{N} \mathcal{E}_n Q_n(t) \\
&+ \mathbb{E}\left[\sum_{n=1}^{N} Q_n(t) \sum_{k=1}^{K} e_{k,n}^t + V \sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)\right] \\
&= B + \mathbb{E}\left[\sum_{n=1}^{N} Q_n(t) \left(\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n\right) | \boldsymbol{Q}(t)\right] \\
&+ V\mathbb{E}\left[\sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)\right] \\
&\overset{\ddagger}{\leq} B + Vl^*
\end{aligned} \tag{25}$$

Since $\Theta(t) \leq B + \mathbb{E}[\sum_{n=1}^{N} Q_n(t)(\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n)|\boldsymbol{Q}(t)] + V\mathbb{E}[\sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)]$, for $\forall t \in \{0, \cdots, T-1\}$ and valid service placement profile $\boldsymbol{\delta}^t$, the inequality still holds when substituting $\boldsymbol{\delta}^t$ by $\boldsymbol{\delta}^{t*}$. Owing to $\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n \leq 0$ for $\boldsymbol{\delta}^{t*}$, the inequality ($\ddagger$) holds.

Take the expectation of the inequality (25) and add it over the time slots $t \in \{0, \cdots, T-1\}$, i.e.,

$$\begin{aligned}
&\sum_{t=0}^{T-1} \mathbb{E}[\triangle(\boldsymbol{Q}(t)) + V\mathbb{E}[\sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)]] \\
&= \sum_{t=0}^{T-1} \mathbb{E}[\mathbb{E}[\mathcal{F}(\boldsymbol{Q}(t+1)) - \mathcal{F}(\boldsymbol{Q}(t))|\boldsymbol{Q}(t)] \\
&+ V\mathbb{E}[\sum_{k=1}^{K} r d_k^t | \boldsymbol{Q}(t)]] \\
&= \sum_{t=0}^{T-1} \mathbb{E}[\mathcal{F}(\boldsymbol{Q}(t+1)) - \mathcal{F}(\boldsymbol{Q}(t))] + V\mathbb{E}[\sum_{k=1}^{K} r d_k^t] \\
&= \mathbb{E}[\mathcal{F}(\boldsymbol{Q}(T)) - \mathcal{F}(\boldsymbol{Q}(0))] + \sum_{t=0}^{T-1} V\mathbb{E}[\sum_{k=1}^{K} r d_k^t] \\
&= \mathbb{E}[\mathcal{F}(\boldsymbol{Q}(T))] + \sum_{t=0}^{T-1} V\mathbb{E}[\sum_{k=1}^{K} r d_k^t] \\
&\leq \sum_{t=0}^{T-1} \mathbb{E}[B + Vl^*] = (B + Vl^*) * T
\end{aligned}$$

Since $\mathbb{E}[\mathcal{F}(\boldsymbol{Q}(T))] \geq 0$, then

$$\sum_{t=0}^{T-1} V\mathbb{E}\left[\sum_{k=1}^{K} rd_k^t\right] \leq (B + Vl^*) * T$$

So we obtain

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\sum_{k=1}^{K} rd_k^t\right] \leq l^* + \frac{B}{V}.$$

Since $\sum_{k=1}^{K} rd_k^t$ is the expected response time for all the services in time slot t, and according to the law of iterated expectations, $\mathbb{E}[\sum_{k=1}^{K} rd_k^t] = \sum_{k=1}^{K} rd_k^t$. Therefore,

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} rd_k^t \leq l^* + \frac{B}{V}.$$

Thus, the service placement profile obtained by solving $\mathcal{P}2$ can infinitely approximate the optimization of $\mathcal{P}1$ by adjusting the variable V. Let $\mathcal{G}(V)$ denote the optimality gap which denotes the difference between the two solutions. The optimal value by solving the supremum of drift-plus-penalty term is $l^* + B/V$, so $\mathcal{G}(V) = B/V$ and $O(\mathcal{G}(V)) = O(1/V)$. Hence, the optimality gap is bounded by $O(1/V)$.

The above analysis can guarantee that the service placement profile obtained by solving $\mathcal{P}2$ can approximately solve $\mathcal{P}1$. However, another issue that needs to be addressed is whether the service placement decisions obtained from $\mathcal{P}2$ do not violate the long-term constraint (11) for $\mathcal{P}1$.

Theorem 2. *The service placement decisions obtained from solving $\mathcal{P}2$ over time slot $t \in \{0, \cdots, T-1\}$ always satisfy the inequality (11).*

Proof. : Let $\boldsymbol{\delta}^{t*}$ denote the service placement decisions of all RSUs $\mathcal{N}$ in time slot t obtained by solving problem $\mathcal{P}2$, i.e., the supremum of drift-plus-penalty term. In other words, when services are placed based on $\boldsymbol{\delta}^{t*}$, the supremum of drift-plus-penalty term can achieve the minimum. For the sake of clear expression and easy discussion, let $Y(\boldsymbol{\delta}^t) = \sum_{k=1}^{K} rd_k^t$ and $Z_n(\boldsymbol{\delta}^t) = \sum_{k=1}^{K} e_{k,n}^t$. We have:

$$\begin{aligned}
&\Theta(t) = \triangle(\boldsymbol{Q}(t)) + V\mathbb{E}[Y(\boldsymbol{\delta}^{t*})|\boldsymbol{Q}(t)] \\
&\leq B - \sum_{n=1}^{N} \mathcal{E}_n Q_n(t) \\
&+ \mathbb{E}[\sum_{n=1}^{N} Q_n(t) Z_n(\boldsymbol{\delta}^{t\ddagger}) + VY(\boldsymbol{\delta}^{t\ddagger})|\boldsymbol{Q}(t)] \\
&\overset{\S}{\leq} B - \sum_{n=1}^{N} \mathcal{E}_n Q_n(t) + \sum_{n=1}^{N} Q_n(t)(\mathcal{E}_n - \epsilon_n) + VL_t^{\ddagger} \\
&= B - \sum_{n=1}^{N} \epsilon_n Q_n(t) + VL_t^{\ddagger},
\end{aligned}$$

where $\boldsymbol{\delta}^{t\ddagger}$ is an arbitrary valid service placement profile for all RSUs $\mathcal{N}$ in time slot t except $\boldsymbol{\delta}^{t*}$ and $L_t^{\ddagger}$ is the corresponding response delay. Due to $\mathbb{E}[Z_n(\boldsymbol{\delta}^{t*})] = Z_n(\boldsymbol{\delta}^{t*}) \leq \mathcal{E}_n$, there exists a small enough $\epsilon_n > 0$ that satisfies $Z_n(\boldsymbol{\delta}^{t*}) \leq \mathcal{E}_n - \epsilon_n$, $\forall n \in \mathcal{N}$, so the inequality (§) holds.

Take the expectations of the above inequality, $\mathbb{E}\{\Theta(t)\} \leq B - \mathbb{E}[\sum_{n=1}^{N} \epsilon_n Q_n(t)] + VL_t^{\ddagger}$, i.e.,

$$\begin{aligned}
&\mathbb{E}[\mathcal{F}(\mathbf{Q}(t+1))] - \mathbb{E}[\mathcal{F}(\mathbf{Q}(t))] + V\mathbb{E}[Y(\boldsymbol{\delta}^{t*})] \\
&\leq B + VL_t^{\ddagger} - \mathbb{E}[\sum_{n=1}^{N} \epsilon_n Q_n(t)]
\end{aligned}$$

By summing this inequality over $t \in \{0, \cdots, T-1\}$, we have

$$\begin{aligned}
&\mathbb{E}[\mathcal{F}(\mathbf{Q}(T))] - \mathbb{E}[\mathcal{F}(\mathbf{Q}(0))] + V\sum_{t=0}^{T-1} \mathbb{E}[Y(\boldsymbol{\delta}^{t*})] \\
&\leq BT + VL_t^{\ddagger}T - \mathbb{E}\left[\sum_{t=0}^{T-1}\sum_{n=1}^{N} \epsilon_n Q_n(t)\right] \\
&\leq BT + VL_t^{\ddagger}T = B'T,
\end{aligned}$$

where $B' = B + VL_t^{\ddagger}$. Since $\mathcal{F}(\mathbf{Q}(T)) = \frac{1}{2}\sum_{n=1}^{N} Q_n^2(T)$, we have $\mathbb{E}[\sum_{n=1}^{N} Q_n^2(T)] \leq 2B'T$. Then, we have,

$$\left(\mathbb{E}\left[\sum_{n=1}^{N} Q_n(T)\right]\right)^2 \leq K\mathbb{E}\left[\sum_{n=1}^{N} Q_n^2(T)\right] \leq 2B'T.$$

Therefore,

$$\frac{1}{T}\mathbb{E}\left[\sum_{n=1}^{N} Q_n(T)\right] \leq \sqrt{\frac{2B'}{T}}.$$

Due to $Q_n(T) \geq 0$ for $\forall n \in \mathcal{N}$, $\mathbb{E}[Q_n(T)]/T \leq \sqrt{2B'}/T$ holds, for $\forall n \in \mathcal{N}$. Given Lemma 1, we have

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n\right] \leq \frac{1}{T}\mathbb{E}[Q_n(T)] \leq \sqrt{\frac{2B'}{T}}.$$

Therefore,

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\sum_{k=1}^{K} e_{k,n}^t - \mathcal{E}_n\right] \leq \lim_{T\to\infty}\sqrt{\frac{2B'}{T}} = 0.$$

Namely,

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} e_{k,n}^t \leq \mathcal{E}_n \quad \forall n \in \mathcal{N}.$$

Therefore, the following conclusions hold: 1) the queue backlog is mean rate stable; and 2) the service placement profile for $\mathcal{P}2$ can solve $\mathcal{P}1$ without violating the constraint (11).

5.2 OPEC Algorithm Description

The algorithm OPEC, which leverages the Lyapunov technology to convert the time slot spanned energy consumption, is shown in Alg. 1. This algorithm aims to obtain the optimal service placement decisions for optimizing problem $\mathcal{P}1$. To this end, the service placement decisions can be obtained by solving the problem $\mathcal{P}2$ along each time slot. In the beginning, some variables should be initialized such as $\mathcal{F}(\boldsymbol{Q}(0))$ and $\mu_{k,n}$. Within each time slot, some variables are observed and calculated such as $\lambda_{k,m}^t$, $\mathcal{R}_{k,m}^t$, and $\lambda_{k,n}^t$. Then the approximately optimum service placement decision is obtained by solving problem $\mathcal{P}2$ (line 8). After that, the algorithm calculates the energy consumption based on the service placement decisions (line 9) and further updates the queue backlog of each queue (line 10).

Algorithm 1. *Online Service Placement for Vehicular Edge Computing (OPEC)*

Require: $Q_n(0)$, $\mathcal{E}_n$, D_n, N, M, T
Ensure: Service placement decisions
1: $\mathcal{F}(\boldsymbol{Q}(0)) = 0$
2: Calculate $\mu_{k,n}$ based on Eq. (6), $\forall k \in \mathcal{K}, \forall n \in \mathcal{N}$
3: **for** $t = 0$ to $T-1$ **do**
4: Observe and record $\lambda_{k,m}^t, \mathcal{R}_{k,m}^t, \forall k \in \mathcal{K}, \forall m \in \mathcal{M}$
5: Calculate $\lambda_{k,n}^t$ based on Eq. (5), $\forall k \in \mathcal{K}, \forall n \in \mathcal{N}$
6: Set $e_{max,n}^t, \forall n \in \mathcal{N}$
7: Predict and obtain $rd_{k,c}^t, \forall k \in \mathcal{K}$
8: Obtain $\boldsymbol{\delta}^{t*}$ by solving problem $\mathcal{P}2$
9: Calculate $e_{k,n}^t$ based on Eq. (9), $\forall k \in \mathcal{K}, \forall n \in \mathcal{N}$
10: Update $Q_n(t+1)$ based on Eq. (16), $\forall n \in \mathcal{N}$
11: **end for**

5.3 Algorithm Design for P2

In algorithm OPEC, there is still a pending issue which needs to be addressed, i.e., the optimization of $\mathcal{P}2$, so in this subsection, we will discuss how to obtain the decision placement decision in each time slot by solving the problem $\mathcal{P}2$ efficiently, i.e., $\boldsymbol{\delta}^{t*} = \arg\min_{\boldsymbol{\delta}^t,\forall t}\{\sum_{n=1}^{N} Q_n(t)\sum_{k=1}^{K} e_{k,n}^t + V\sum_{k=1}^{K} rd_k^t\}$.

Lemma 3. *The optimization problem $\mathcal{P}2$ is* NP-Hard.

Proof. : It is well-known that the multiple knapsack problem (MKP) is an NP-hard problem [25], and it can be formally described as follows. Given a set of knapsacks $\mathcal{G}$ and items $\mathcal{I}$, respectively, each knapsack $k \in \mathcal{G}$ can pack finite items based on its knapsack capacity a_k and each item $i \in \mathcal{I}$ has a size s_i and profit p_i. MKP aims to find a way to assign items into the knapsacks such that the total profits can be maximized, i.e.,

$$(\mathcal{P}3) \quad \max \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} x_{k,i}p_i$$

$$\text{s.t.}: \quad \sum_{i\in\mathcal{I}} x_{k,i}s_i \leq a_k \quad \forall k \in \mathcal{G}, \tag{26}$$

$$x_{k,i} \in \{0,1\} \quad \forall k \in \mathcal{G}, \forall i \in \mathcal{I}. \tag{27}$$

In the next, we will prove that our problem is actually an instance of MKP, i.e., $\mathcal{P}2$ is equivalent to $\mathcal{P}3$. First,

it is easy to transform $\mathcal{P}2$ into a maximization problem, e.g., by the inverse operation. Then, given our system model, the set of RSUs $\mathcal{N}$ and the set of services $\mathcal{K}$ correspond to the set of knapsacks $\mathcal{G}$ and items $\mathcal{I}$, respectively. Each service has the required amount of task-input data (e.g., d_k) which corresponds to the item size s_i. Each RSU has a storage capacity D_n that corresponds to the backpack capacity a_k. The total amount of task-input data for all the tasks assigned to the RSU should not exceed its storage capacity. Such a constraint is actually the knapsack constraint in Eq. (26). The objective function can be considered as the profit of each service in $\mathcal{P}2$. Therefore, $\mathcal{P}2$ is equivalent to $\mathcal{P}3$ where K items are packed to N RSUs to optimize the response delay. Accordingly, the optimization problem $\mathcal{P}2$ is *NP-Hard*.

The searching space in the potential solution domain is up to K^N, which thus prohibits the exhaustive search in reality, especially considering the strict latency requirements of vehicular services. In addition, the iteration-based evolutionary algorithms have similar drawbacks in handling the strict latency requirements. Indeed, the time taken to make decisions on service placement at RSUs should be as fast as possible. Thus, we in this paper put forward a greedy algorithm for this problem to cater to such requirements in the next.

Actually, several factors could affect the performance of the service placement decision in pursuit of energy consumption and response latency optimization. For instance, the request rate of service usually indicates its popularity. Intuitively, to place the most popular services at RSUs can reduce the response delay to the most extent. Further to this, services with larger size of task-input data tend to be placed at RSUs, since they actually consume more network resources than those with smaller task-input data size. In the next, services that require less computational resources tend to be placed, since services with larger computational demands usually incur relatively long computation delay and large energy consumption. Last but not least, the value of V can also affect the objective value of $\mathcal{P}2$. V is used to adjust the tradeoff between response latency and energy consumption. Usually, a larger value of V indicates that the VEC system values the response latency more than the energy consumption, while a smaller value of V indicates that the system values energy consumption more due to the limited energy supply. Specifically, we use the average service requests arriving at RSU n for service k denoted by $p_{k,n}^t$ as the popularity of service k, and $p_{k,n}^t$ is defined as:

$$p_{k,n}^t = \sum_{m=1}^{M} I_{n,m} \frac{\lambda_{k,m}^t}{|\mathcal{R}_m|}, \quad \forall k \in \mathcal{K}, \forall n \in \mathcal{N}. \tag{28}$$

Based on these observations, we design a utility function that incorporates the above three factors into one single value to evaluate each service within each time slot. In particular, the utility function of service k at RSU n in time slot t can be defined as:

$$\begin{aligned}\mathcal{U}_{k,n}(t) = w_p &\frac{p_{k,n}^t - \min\limits_{l\in\mathcal{K}}(p_{l,n}^t)}{\max\limits_{l\in\mathcal{K}}(p_{l,n}^t) - \min\limits_{l\in\mathcal{K}}(p_{l,n}^t)} \\ + w_d &\frac{d_k - \min\limits_{l\in\mathcal{K}}(d_l)}{\max\limits_{l\in\mathcal{K}}(d_l) - \min\limits_{l\in\mathcal{K}}(d_l)} \\ + w_s &\frac{\max\limits_{l\in\mathcal{K}}(s_l) - s_k}{\max\limits_{l\in\mathcal{K}}(s_l) - \min\limits_{l\in\mathcal{K}}(s_l)} + w_v \frac{V_{max} - V}{V_{max} - V_{min}},\end{aligned} \tag{29}$$

where w_p, w_d, w_s and w_v are numerical values to denote the preferences towards the four factors which satisfy $w_p + w_d + w_s + w_v = 1$. The two functions $\max(\cdot)$ and $\min(\cdot)$ denote the maximal and minimal values, respectively. V_{max} and V_{min} are the upper and lower bounds of V, respectively. Therefore, the service with a larger utility value is more worth being placed at RSUs.

A greedy algorithm GAPO for solving $\mathcal{P}2$ is put forward as depicted in Algorithm 2. As illustrated in our system model, RSUs are inter-connected with each other, so the information collected from regions such as $\lambda_{k,m}^t$ can be timely disseminated and shared among them. Some initializations should be accomplished such as $\boldsymbol{\delta}^t = \mathbf{0}$. Additionally, the algorithm initializes one Max-Heap h_n for each RSU n within each time slot. In the next, each RSU can make their own service placement decisions independently. Specifically, RSU n calculates the utility value of each service based on the gathered information. The heuristic rule is that services with larger utility values tend to be placed at RSUs. To achieve this goal, the al-

gorithm pushes the pair of service index and the corresponding utility value (i.e., $(k, \mathcal{U}_{k,n}^t)$) into the heap h_n. Thus, service with a larger utility value is always retrieved first when the pop operation over h_n is implemented. Then n checks whether there is enough storage capacity for the current service popped from h_n. If the storage capacity is sufficient, the current service is placed and the service placement decision is updated (line 15). The procedure repeats until h_n is empty or there is not enough storage capacity for RSU n.

Algorithm 2. *Greedy Algorithm for Per Slot Optimization (GAPO)*

Require: $\mathcal{K}, \mathcal{N}, V, \mathcal{E}_n, D_n, w_p, w_d, w_s, t$
Ensure: $\boldsymbol{\delta}^t$
1: Record and share $\lambda_{k,m}^t$ among RSUs $\mathcal{N}$, $\forall k \in \mathcal{K}$, $\forall m \in \mathcal{M}$
2: Initialize $\boldsymbol{\delta}^t$ with zeros
3: Initialize a heap h_n for each $n \in \mathcal{N}$
4: **for** each n in $\mathcal{N}$ **do**
5: $Csum = 0$
6: **for** each k in $\mathcal{K}$ **do**
7: Calculate $p_{k,n}^t$ based on Eq.(28)
8: Calculate $\mathcal{U}_{k,n}^t$ based on Eq.(29)
9: Push $(k, \mathcal{U}_{k,n}^t)$ into h_n
10: **end for**
11: **while** h_n not empty **do**
12: $h_n.pop(l, \mathcal{U}_{l,n}^t)$
13: $Csum = Csum + d_l$
14: **if** $Csum \leq D_n$ **then**
15: $\delta_{l,n}^t = 1$
16: **end if**
17: **end while**
18: **end for**
19: Return $\boldsymbol{\delta}^t$

Remark: GAPO is actually executable in parallel to a great extent after each RSU n obtains the required information from the VEC network. In GAPO, the time taken for each RSU n mainly includes the following parts. The first part is the time taken for RSUs to communicate with each other such that the information required for decision making can be gathered. The second part is the time taken to sort the services in h_n and the process can be accomplished with the time complexity $O(K \log K)$, where K is the number of services. The third part is the time taken to traverse h_n with the aim to find appropriate services to place, and the process can be accomplished with the worst time complexity $O(K \log K)$. Note that the first part of time is usually negligible compared to the other two parts. To sum up, the time complexity of GAPO is $O(NK \log K)$ without considering the parallel execution of each RSU. The time complexity is much less than the evolutionary algorithms such as the Genetic Algorithm (GA).

VI. SIMULATION EVALUATION

We have conducted a series of experiments to evaluate the placement strategy in this section. In the next, we will give parameter settings in the simulation, and discuss the simulation results, respectively.

6.1 Parameter Settings

The main parameters with the default values in the simulation are listed in Table 1. Note that the parameters are set experientially. For example, we assume that the topology of the vehicular edge computing network is built in a random way. In particular, the set of RSUs which can cover the region m (i.e., $\mathcal{R}_m$) is determined randomly. Additionally, we assume that the transmission delay can be predicted when task data are transmitted over the backbone network and thus the response delay for task execution in the cloud center can be obtained. For the utility function of services at RSUs, the weights towards the four factors are set to 0.3, 0.2, 0.2 and 0.3, respectively.

6.2 Results Analysis and Discussion

Figure 2 shows the performance comparison in terms of the optimal values. In addition to the proposed heuristic, there are actually several approaches for solving the problem $\mathcal{P}2$. In particular, we compare our approach with three other approaches in the simulation. The three approaches are the greedy approach, the random approach and the genetic approach, respectively. We denote them by P_Max, RND, and GA in the simulation, respectively. The greedy approach only adopts the service popularity as the selection criteria. Within each time slot, the services are sorted in descending order based on the defined service popularity. Then, the service with the largest value of popularity is always placed first. The procedure repeats

Table 1. *Parameter Settings*

Parameter	Descriptions	Value
M	The number of regions	6
N	The number of RSUs	20
K	The number of services	[25, 65]
$\lambda_{k,m}^t$	Arrival rate of requests for service k in region m	[10, 20]
$\mathcal{E}_n$	Energy consumption constraint for RSU n	[1.2, 2.2]
f_n	The processing frequency	[5MHz, 1GHz]
d_k	The task-input data size for service k	[1, 6]
s_k	The averagely required workload for service k	[20, 25]
D_n	The caching capability of RSU n	[30, 50]
$\varsigma\eta$	The coefficients for the energy consumption	1e-12
α_n	The static energy consumption	[0, 1)

until the violation of constraint conditions such as the inequation (1) happens. The random approach is the simplest way to place the services for RSUs within each time slot. During each time slot, services are randomly selected to place as long as no violation happens. The random approach does not consider the feature of the optimization objective or the feature of service requests. The genetic algorithm is especially suitable for binary MKP, owing to its simple deployment and easy implementation. However, the solution accuracy of this approach mainly depends upon the number of iterations, which often turns out to be a time-consuming process. Therefore, it can hardly satisfy the stringent latency requirement in this paper.

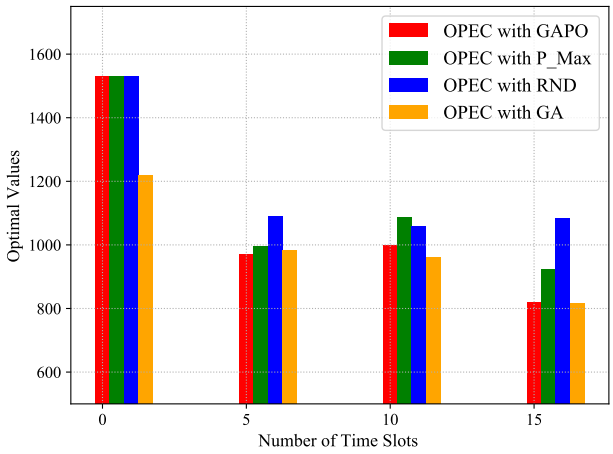

Figure 2. *The performance comparison with different approaches for $\mathcal{P}2$.*

From the figure, we can observe that the average performance of OPEC with GA is the best among the four approaches, and OPEC with GAPO comes second. In the parameter settings for GA, the crossover and mutation probability are respectively set to 0.2 and 0.02. The population size and the maximal number of iterations are 20 and 30, respectively. In the simulation, some information is the same for all the time slots, which includes M, N, K, $\mathcal{R}_m$, $\mathcal{E}_n$ and so on. To reflect the dynamics of the vehicular edge computing networks, some information is totally independent in each time slot. Such information is exemplified by $\lambda_{k,m}^t$, d_k and s_k.

As a result, we can easily observe that the performance of all approaches wildly fluctuates in different time slots. Even so, our approach is averagely much better than the random approach and the greedy approach. The random approach can occasionally yield better results, e.g., when the number of time slots is 10. In contrast, the greedy approach, which only considers the service popularity, never yields the best result but occasionally yields the worst result as denoted in the figure. This is because it neglects two important factors such as d_k and s_k, which can seriously influence the value of the optimization objective. For example, based on Eq. (9), it is obvious that the service with a larger value of s_k renders more energy consumption if it is placed at RSU. However, energy consumption has been incorporated into the optimization of objection function as shown in problem $\mathcal{P}2$. Therefore, minimization of the optimization objective is supposed to consider the influence of the averagely required work-

load s_k. It shall be noted that the performance of OPEC with GAPO, P_Max, RND is equally bad compared to OPEC with GA, when the time slot is zero. The service placement profile is generated randomly for the three approaches at the beginning. However, this initial service placement profile can be updated with the increasing iterations in OPEC with GA.

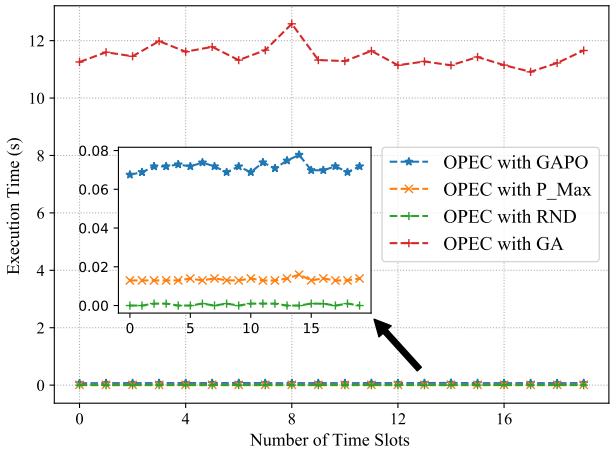

Figure 3. *The execution time comparison with different approaches for $\mathcal{P}2$.*

The execution times for different approaches are depicted in Fig. 3. Comparatively speaking, the average execution time of OPEC with GAPO is slightly longer than the other two approaches, i.e., OPEC with RND and OPEC with P_Max. However, these three approaches can almost achieve real-time response in the simulation. Obviously, the execution time of OPEC with GA is much longer than the above three approaches. As shown in 3, this strategy usually takes seconds to get the result in the simulation and thus the execution times of other three approaches are negligible compared to that of OPEC with GA. Combining with the results in Fig. 2, we can see that GA helps OPEC achieve the best performance in terms of optimal values at the expense of unbearable time overhead. Considering the strict delay requirement for obtaining the optimal service placement profile, GA based approach is not appropriate in this paper.

In the next, we evaluate the average energy consumption at different RSUs along the time slots. The experimental results are shown in Fig. 4. On one hand, we need to investigate the energy consumption of each RSU along the time slots; On the other hand, it is more important to check whether the violation happens along the time slots. Specifically, we select 4 from 20 RSUs in the simulation, and the label "RSU: 1.27" means the RSU with its energy threshold being 1.27. First, it is noticeable that the energy consumption of the four RSUs do not exceed their own global energy constraints. As a result, the simulation result has verified the rationality of the theoretical analysis. Second, the energy consumption for RSUs all seem stable along the time slots. Such an observation is acceptable and understandable, since it is in accord with the conclusion drawn in Theorem 2, i.e., the queue backlog is mean rate stable. Last but not least, we notice that the energy consumption for all the RSUs are the least when the time slot is zero. It is explicable since we assume that there are no services to be placed at RSUs in the simulation. Thus, there are no services at RSUs which incur energy consumption at the initial time slot.

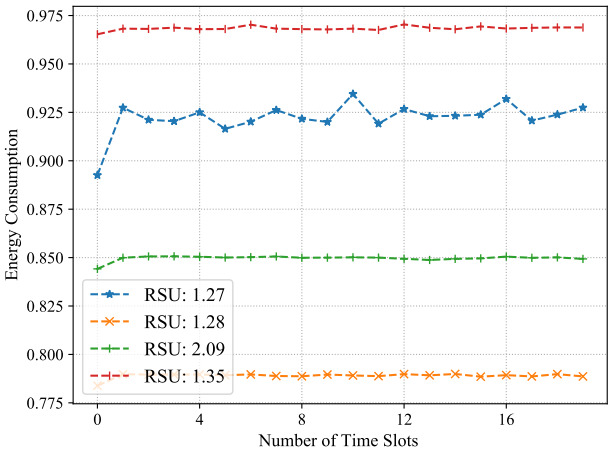

Figure 4. *The energy consumption along different time slots.*

We investigate the influence of the number of services upon the performance of our approach in the next. The simulation results are shown in Fig. 5. Specifically, the number of services varies from 20 to 65 with a step of 5 in the simulation. Fig. 5 reveals the optimal values with different numbers of services. Several conclusions can be drawn from the observation as follows. First, it seems that the optimal value of the optimization objective has little to do with the number of services, since there are no deterministic relationships between them. For example, when the number of services is 25, the optimal value in most cases is better than that with the number of services

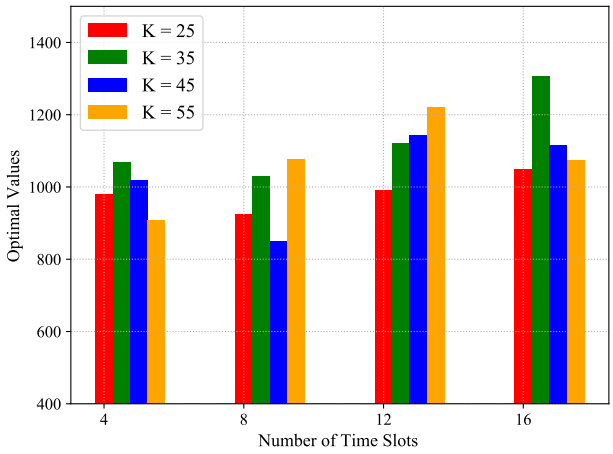

Figure 5. *The performance comparison under different numbers of services.*

equal to 35 and 45. On the other hand, when the number of services is 55, the optimal value in most cases is also better than that with the number of services equal to 35 and 45, respectively. Accordingly, the number of services is not the more the better, or the less the better, in terms of the optimal values.

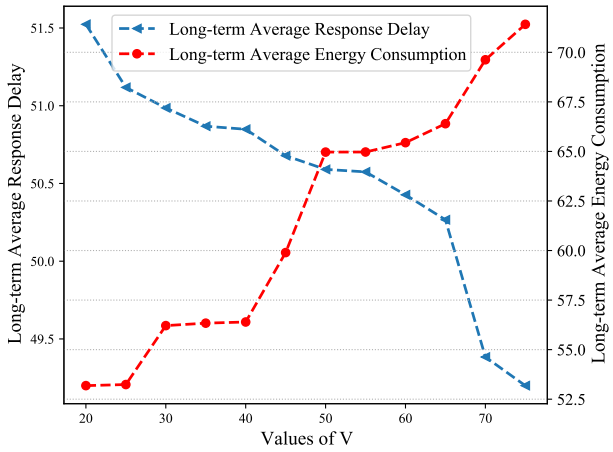

Figure 6. *The performance comparison with different values of V.*

The control parameter V can be used for making a tradeoff between the response latency and energy consumption as mentioned earlier. We study the influence of V upon the long-term average response delay and the long-term average energy consumption in the simulation. The results are shown in Fig. 6. In particular, the value of V varies from 20 to 80 in the simulation. Obviously, the long-term average energy consumption for all RSUs increases with the increasing value of V. In contrast, the long-term average response delay decreases as V increases. According to the theoretical analysis, the placement decision profile obtained by the proposed algorithm can gradually approach the best decision profile by increasing the value of V. However, it is inadvisable to blindly increase the value of V, considering the ever-increasing average energy consumption for all the RSUs in the long run. It is better to seek a tradeoff between the response delay and energy consumption when V is set in the simulation.

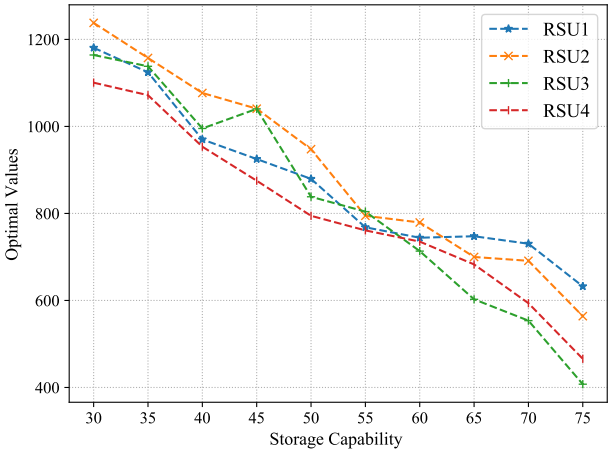

Figure 7. *The performance comparison with different storage capabilities.*

We investigate the influence of the storage capabilities upon the performance of OPEC. The simulation results are shown in Fig. 7. We still select 4 from 20 RSUs as the observation subjects in the simulation. Obviously, greater storage capabilities enable more services to be placed at the same time. However, the more the services to be placed, the more the energy consumption at RSUs. Thus, it is worthwhile investigating the relationships between the performance of the approach and the storage capacity. The simulation result in the figure shows that the performance of OPEC is constrained by the storage capacity at RSUs. For instance, greater storage capacity yields better performance in terms of the optimal values.

Since our goal in this paper is to reduce the average response latency for all the requested services by placing services at the edge in advance, we have conducted the last experiment to evaluate our approach compared to the traditional approach that does not ap-

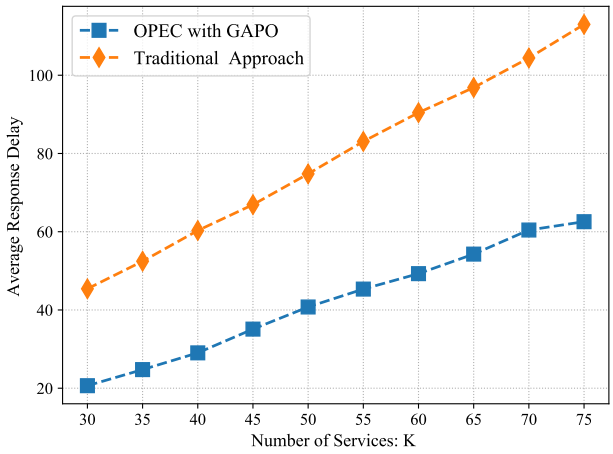

Figure 8. *The response delays with different number of services.*

ply service placement strategy. In this simulation, the number of services varies from 35 to 75 with a step 5, and the simulation result is shown in Fig. 8. It is obvious that a better result is yielded when the service placement strategy is applied. In addition, the response delays increase for both approaches as the number of services increases. More important, our approach is much better than the traditional approach no matter how the number of services varies.

VII. CONCLUSION

VEC can cater to the strict delay requirement of vehicular tasks/services, since the computing resources are deployed at the network edge such as RSUs. However, the explosive growth in vehicular offloading requests has caused tremendous pressure on both the edge server and the front-haul links. In this paper, we attempt to optimize the VEC systems by service placement strategy. Specifically, we minimize the average response latency for the requested services along the slotted timeline, while considering multiple constraints such as energy consumption and storage capacity. The greedy heuristic has been incorporated into the drift-plus-penalty based algorithm for searching the approximate solution. Extensive simulation has proven that our approach can achieve better performance compared with other approaches in terms of optimal values.

References

[1] LI X, ZHANG X, HUANG T. Asynchronous online service placement and task offloading for mobile edge computing[C]//18th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2021, Rome, Italy, July 6–9, 2021. [S.l.]: IEEE, 2021: 1-9.

[2] XU J, CHEN L, ZHOU P. Joint service caching and task offloading for mobile edge computing in dense networks[C]//2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018. [S.l.]: IEEE, 2018: 207-215.

[3] TANG C, WEI X, ZHU C, et al. Mobile vehicles as fog nodes for latency optimization in smart cities[J]. IEEE Trans. Veh. Technol., 2020, 69 (9):9364-9375.

[4] CHEN L, SHEN C, ZHOU P, et al. Collaborative service placement for edge computing in dense small cell networks[J]. IEEE Trans. Mob. Comput., 2021, 20(2):377-390.

[5] TANG C, ZHU C, WU H, et al. Towards response time minimization considering energy consumption in caching assisted vehicular edge computing[J/OL]. IEEE Internet of Things Journal, 2021:1-1. DOI: 10.1109/JIOT.2021.3108902.

[6] TANG C, ZHU C, WU H, et al. Caching assisted correlated task offloading for iot devices in mobile edge computing[C]//IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021. [S.l.]: IEEE, 2021: 1-6.

[7] XIE R, TANG Q, QIAO S, et al. When serverless computing meets edge computing: Architecture, challenges, and open issues[J]. IEEE Wirel. Commun., 2021, 28(5):126-133.

[8] HOU H, JIN H, LIAO X. Cost efficient edge service placement for crowdsensing via bus passengers[J]. Mob. Networks Appl., 2021, 26(2):899-908.

[9] AYOUBI M, RAMEZANPOUR M, KHORSAND R. An autonomous iot service placement methodology in fog computing[J]. Softw. Pract. Exp., 2021, 51(5):1097-1120.

[10] HUANG T, LIN W, XIONG C, et al. An ant colony optimization-based multiobjective ser-

vice replicas placement strategy for fog computing[J]. IEEE Trans. Cybern., 2021, 51(11):5595-5608.

[11] XU X, SHEN B, YIN X, et al. Edge server quantification and placement for offloading social media services in industrial cognitive iov[J]. IEEE Trans. Ind. Informatics, 2021, 17(4):2910-2918.

[12] HAO Y, CHEN M, GHARAVI H, et al. Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system [J]. IEEE Trans. Ind. Informatics, 2021, 17(8): 5552-5561.

[13] CHEN Y, ZHANG S, JIN Y, et al. LOCUS: user-perceived delay-aware service placement and user allocation in MEC environment[J]. IEEE Trans. Parallel Distributed Syst., 2022, 33(7): 1581-1592.

[14] ORTÍN J, GÁLLEGO J R, HERNÁNDEZ-SOLANA Á, et al. On optimizing network function placement for multicast group call service provision in LTE IOPS networks[J]. IEEE Access, 2021, 9:160897-160916.

[15] YUAN B, GUO S, WANG Q. Joint service placement and request routing in mobile edge computing[J]. Ad Hoc Networks, 2021, 120: 102543.

[16] PANADERO J, SELIMI M, CALVET L, et al. A two-stage multi-criteria optimization method for service placement in decentralized edge microclouds[J]. Future Gener. Comput. Syst., 2021, 121:90-105.

[17] HAN P, LIU Y, GUO L. Interference-aware online multicomponent service placement in edge cloud networks and its AI application[J]. IEEE Internet Things J., 2021, 8(13):10557-10572.

[18] FARHADI V, MEHMETI F, HE T, et al. Service placement and request scheduling for data-intensive applications in edge clouds[J]. IEEE/ACM Trans. Netw., 2021, 29(2):779-792.

[19] NING Z, DONG P, WANG X, et al. Distributed and dynamic service placement in pervasive edge computing networks[J]. IEEE Trans. Parallel Distributed Syst., 2021, 32(6):1277-1292.

[20] HE X, JIN R, DAI H. Joint service placement and resource allocation for multi-uav collaborative edge computing[C]//IEEE Wireless Communications and Networking Conference, WCNC 2021, Nanjing, China, March 29 - April 1, 2021. [S.l.]: IEEE, 2021: 1-6.

[21] LIU Z, ZHAN C, CUI Y, et al. Robust edge computing in uav systems via scalable computing and cooperative computing[J]. IEEE Wireless Communications, 2021, 28(5):36-42.

[22] TANG C, ZHU C, WEI X, et al. Integration of UAV and fog-enabled vehicle: Application in post-disaster relief[C]//25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019. [S.l.]: IEEE, 2019: 548-555.

[23] GEORGIADIS L, NEELY M J, TASSIULAS L. Resource allocation and cross-layer control in wireless networks[J]. Foundations & Trends in Networking, 2006, 1(1):1-144.

[24] NEELY, MICHAEL J. Stochastic network optimization with application to communication and queueing systems[J]. Synthesis Lectures on Communication Networks, 2010, 3(1):211.

[25] KORTE B, VYGEN J. Combinatorial optimization: Theory and algorithms[M]. 1st ed. [S.l.]: Springer, 2000.