Toward Response Time Minimization Considering Energy Consumption in Caching-Assisted Vehicular Edge Computing

Chaogang Tang, *Member, IEEE*, Chunsheng Zhu, *Member, IEEE*, Huaming Wu, *Member, IEEE*, Qing Li, *Senior Member, IEEE*, and Joel J. P. C. Rodrigues, *Fellow, IEEE*

***Abstract*—The advent of vehicular edge computing (VEC) has generated enormous attention in recent years. It pushes the computational resources in close proximity to the data sources and thus, caters for the explosive growth of vehicular applications. Owing to the high mobility of vehicles, these applications are of latency-sensitive requirements in most cases. Accordingly, such requirements still pose a great challenge to the computing capabilities of VEC, when these applications are outsourced and executed in VEC. Against this backdrop, we propose a new mathematical model, which, respectively, generalizes the computation and communication models, and applies application-oriented caching into VEC in this article. Based on this model, a new strategy is further proposed to optimize the average response time of applications over an infinite time-slotted horizon for VEC. A long-term energy consumption constraint is imposed to guarantee the stability of the VEC system, and the Lyapunov optimization technology is adopted to tackle this constraint issue. Two greedy heuristics are put forward to help find the approximate optimal solution in the drift-plus-penalty-based algorithm. Extensive experiments have been conducted to evaluate the response time and energy consumption in the caching-assisted VEC. The simulation results have shown that the proposed strategy can dramatically optimize the average response time while satisfying the long-term energy consumption constraint.**

***Index Terms*—Caching, greedy heuristics, Lyapunov optimization, service provisioning, vehicular edge computing (VEC).**

I. Introduction

THE RAPID development of information and communication technology gives rise to the prosperities of smart vehicles [1], [2]. In addition to the ability to communicate with each other, smart vehicles are also equipped with computer facilities to perform vehicular applications. However, a large number of vehicular applications have created enormous pressure on such "computers with the wheels." On the other hand, the advent of vehicular edge computing (VEC) [3] has generated enormous attention recently. In comparison with the limited capabilities of the onboard computers, there are more computational resources in VEC. Furthermore, VEC pushes these resources in close proximity to the data sources. For instance, the edge servers in VEC are usually deployed at the logical edge of networks, e.g., roadside unit (RSU). Vehicles can outsource their applications to RSU via vehicle-to-infrastructure (V2I) communication technologies. Hence, this computing paradigm caters for the increasingly complicated requirements of vehicular applications.

Additionally, VEC can alleviate traffic congestion in the network core and lays profound foundation for the development of vehicular applications in smart transportation. Despite the benefits brought for vehicular applications, VEC is still confronted with a big challenge. Most of the vehicular applications are time sensitive such as route navigating [4], owing to the high mobility of vehicles. However, it is very challenging for VEC to satisfy the strict latency requirements, due to the fact that the computational resources in VEC are not inexhaustible compared to cloud computing. The response latency reduction comes at the expense of the computational resource constraint in VEC.

Therefore, significant efforts are still needed to improve the performance of VEC systems. For this purpose, we propose to combine the application-/service-oriented caching with VEC in this article. By caching the most frequently outsourced applications in VEC, the response time can be

Manuscript received April 17, 2021; revised August 7, 2021; accepted August 21, 2021. Date of publication August 30, 2021; date of current version March 24, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB2104301; in part by the Chinese National Research Fund (NSFC) Key Project under Grant 61532013 and Grant 61872239; the Project "Network Communication Intelligent Core Chip Design and Core Software (PCL2021-A08)," and the Project "Beihang Beidou Technological Achievements Transformation and Industrialization Funds (BARI2005)"; in part by the FCT/MCTES through national funds and when applicable co-funded EU Funds under Project UIDB/50008/2020; and in part by the Brazilian National Council for Research and Development (CNPq) under Grant 313036/2020-9. *(Corresponding author: Chunsheng Zhu.)*

Chaogang Tang is with the School of Computer Science and Technology and the Mine Digitization Engineering Research Center of the Ministry of Education, China University of Mining and Technology, Xuzhou 221116, China (e-mail: cgtang@cumt.edu.cn).

Chunsheng Zhu is with the SUSTech Institute of Future Networks, Southern University of Science and Technology, Shenzhen 518055, China, and also with the PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen 518055, China (e-mail: chunsheng.tom.zhu@gmail.com).

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn)

Qing Li is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: qing-prof.li@polyu.edu.hk).

Joel J. P. C. Rodrigues is with the Federal University of Piauí, Teresina 64049-550, Brazil, and also with the Instituto de Telecomunicações, 3810-193 Aveiro, Portugal (e-mail: joeljr@ieee.org).

Digital Object Identifier 10.1109/JIOT.2021.3108902

tremendously reduced. In our view, caching-assisted VEC can greatly improve the efficiency of application outsourcing and task performing, and thus, fulfill the key requirements of the computationally intensive and time-sensitive applications. However, different from content delivery in caching-enabled information centric networks [5], [6], application outsourcing in the caching-assisted VEC is much more complicated in terms of response time and energy consumption [7]–[10].

For example, virtual machine (VM) and lightweight container Docker [11] are the two most widely used frameworks in the provision of application outsourcing at the edge. Both of them will temporarily initialize virtual environments for the offloaded applications. If the applications are cached at the server, it will be better to maintain the running state of virtual environments for a certain period of time, with the aim to avoid additional time overheads on virtual initialization. This way, however, will incur additional energy consumptions. Since the energy consumption can serve as an important performance indicator to evaluate the VEC systems, the trade-off between the energy consumption and application caching strategies should be carefully considered. On another hand, the evaluation of the caching-assisted VEC system, such as the response time optimization, usually needs a long-term process. Therefore, it becomes significant and yet very challenging to design an appropriate application/service caching strategy in VEC.

In this article, we focus on the response time minimization while considering the energy consumption for application outsourcing in the caching-assisted VEC. In particular, we list the major contributions of this article as follows.

1) We propose a generic approach to improve the performance of application outsourcing in the caching-assisted VEC. Specifically, we mathematically formulate the optimization problem, with the aim to minimize the average response time of the outsourced applications over a long time-slotted horizon in this article.
2) Owing to the difficulty of solving the optimization problem while satisfying the energy consumption over an infinite time-slotted horizon, we introduce the Lyapunov optimization technology in this article. By doing so, the long-term energy constraint can be converted into per-slot ones. Furthermore, a dynamic algorithm based on the *drift-plus-penalty* term is proposed to obtain the approximate optimal solution over the accumulated time slots.
3) Extensive experiments are carried out to evaluate the response time and energy consumption of the proposed caching strategy in VEC. Simulation results show that the proposed application outsourcing strategy in the caching-assisted VEC can achieve better performance while satisfying the time slot spanned energy constraint.

The remainder of this article is organized as follows. Representative works are studied and discussed in Section II. In Section III, we apply application-oriented caching into VEC and formally establish a mathematical model to minimize the response time while considering the energy consumption in the caching-assisted VEC. In Section IV, we apply the Lyapunov optimization technology to tackle the time slot spanned energy constraint, and further put forward a drift-plus-penalty-based algorithm that leverages two greedy heuristics to find the approximate optimal solution in the caching-assisted VEC. The simulation results are reported and discussed in Section V, followed by a conclusion in Section VI.

II. Related Works

With the advent of smart transportation and its corresponding subecosystems, such as smart vehicle and RSU, the vehicle-loaded computers have gained rapid development. For example, the computational resources can be extended to such entities, enabling various applications and services running at the vehicles and RSUs, while satisfying the strict response time requirement. VEC has been considered as a promising and efficient approach to relieve the burden of backbone networks [12], because a vast amount of data generated by numerous smart terminals can be processed locally or at the logical edge of networks, with no need for task offloading to the remote cloud center via the core networks.

Recently, the task offloading and service outsourcing in VEC have indeed attracted increasing attention by virtue of its advantages [13]–[15]. We plan to investigate the existing literature on VEC from two aspects in what follows.

A. Caching-Based VEC

Inspired by the content-centric mobile edge caching, researchers turn their attention to the cache-enabled task offloading in VEC [7], [8], with a purpose of further optimizing energy consumption, response time, and so on. For instance, if the tasks are cached in VEC, vehicles do not need to offload the tasks any longer and hence, the transmission delay and execution time in the edge can be omitted [9], [10].

Xu *et al.* [7] suggested that a small number of services can be cached in size-limited edge server so as to improve QoS with regards to (w.r.t.) response time. To investigate the performance of the caching strategies, the optimization problem is formulated to minimize the energy consumption in the long run, i.e., across different time slots. An efficient online algorithm is proposed to tackle these issues in mobile-edge computing systems, which tries to jointly optimize the dynamic service caching and task offloading.

The computation-intensive and time-sensitive tasks are becoming dominating recently, which benefits from increasingly popular applications, such as augment reality and in-car gaming. Hao *et al.* [8] considered the task caching as a promising solution to satisfy the low-latency requirement. In this regard, they propose to cache the completed task application and corresponding data in the edge cloud, and try to jointly optimize the task caching and offloading with multiple constraints. An alternating iterative algorithm is proposed to tackle this problem and simulation results prove that it indeed outstands other approaches.

To realize efficient information dissemination, Qin *et al.* [16] constructed a hierarchical end-edge framework, which combines data delivery, computation offloading, and content caching. The network overhead is selected as the performance indicator to evaluate this framework and the

related strategy. In particular, they model the optimization problem as a mixed-integer nonlinear programming problem and solve it based on the deep deterministic policy gradient. Simulation results have revealed its advantages compared to other benchmark approaches.

A mass of vehicles can serve as the intermediate nodes to provide caching services for the requests in proximity. However, several issues should be addressed, which include the QoS, incentives, and privacy. To this end, Dai *et al.* [17] combined the deep reinforcement learning with the permissioned blockchain to achieve an intelligent and secure content caching in the VEC system and network. Simulation results have indicated that the approach is much better than two other benchmark methods.

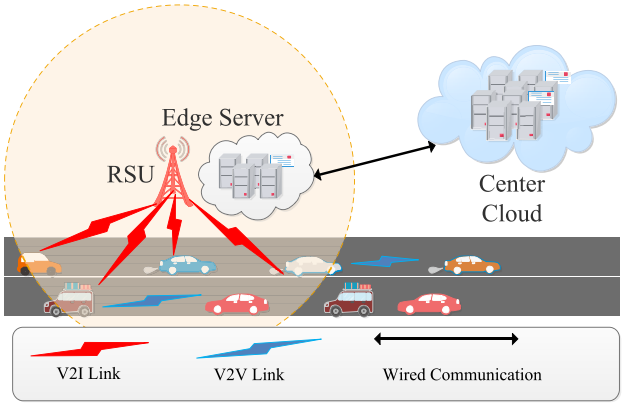


Fig. 1. Application scenario.

B. Response Time and Energy Consumption About VEC

Vehicles can serve as the computing nodes to provide the computational resources to cater for the time-sensitive applications in the vicinity [8], [18]–[20]. For example, with the increasing number of applications, various computational entities are welcome with the aim to mitigate the high demands of computing resources in smart city. Hou *et al.* [18] first proposed that vehicles with idle computing resources should act as the infrastructures to perform multiple functionalities such that vehicles can play more important roles in communication and computation. Specifically, they have discussed several kinds of communication and computational infrastructures where either moving or parked vehicles can serve as the computing entities for response latency optimization or improving the throughput of the computing paradigm.

Similar works can also be found in [19] where authors suggest that the ride-share taxis can serve as the infrastructure to provide both communication and computation resources for the time-sensitive applications. To be specific, terminal users can leverage these taxis to support live video streaming, e.g., taxis can proactively receive video chunks. Furthermore, they formulate the problem as a coverage optimization problem and solve it by the strategy that is applied to the set cover problem (SCP). Simulation results have proven its advantages compared to other strategies.

On the other hand, the resource-hungry and time-sensitive in-car applications have put huge pressure on the limited computing capabilities of vehicles. Meanwhile, VEC has shown great potentials to tackle such an issue, e.g., by outsourcing the vehicular applications to the ubiquitous vehicular edge servers such as RSU [14], [20], [21].

A multiuser multiserver VEC system is considered in [21] where the load balance-aware resource allocation is studied when tasks are offloaded. Authors have regarded this problem as a mixed-integer nonlinear programming problem for the system utility optimization. In particular, they divide it into two subproblems and design efficient algorithms for them.

Ning *et al.* [22] strived to establish an offloading system in VEC with the help of a deep reinforcement learning technology. Specifically, the finite Markov chains are applied to modeling the communication and computation states. The optimization problem is modeled mathematically in the hope to improve the Quality of Experience (QoE) of users. Similar to [21], the problem is also decoupled into two suboptimization problems, and solved by a low-complexity algorithm.

Owing to the resource-hungry feature of the tasks, a certain portion of tasks pertaining to the strict latency requirement is encouraged to offload to the external entities for execution. VEC can play an important role in assisting such kinds of task execution. To predict the resource consumption in the edge nodes more accurately, Chen *et al.* [23] tried to achieve an adaptive selection of machine learning algorithms, by using a two-stage metalearning-based approach and extracting metafeatures on the database. Multiple offloading decisions are considered in [15] where tasks are offered three options for the execution, i.e., local host, cloudlet, and cloud center. To achieve better performance of the cloudlet-cloud offloading, a mathematical problem is proposed that aims to optimize the energy consumption in the long run.

For the same task with different task-input parameters, the computational result could be of great difference. Accordingly, the cached computational result may not benefit the following requestors even if the tasks are the same. Few of the aforementioned works have considered this case. Compared to these works, we, respectively, generalize the computation, communication, and caching model to accommodate this case, and aim to minimize the response time with the consideration of the energy consumption constraint in the caching-assisted VEC.

III. Proposed Caching-Assisted VEC Model

We consider a VEC scenario consisting of one edge server S and multiple moving vehicles, as shown in Fig. 1. The edge server can be deployed at a RSU, which directly connects to the cloud server on the one hand, and communicates with nearby vehicles using the vehicle-to-infrastructure (V2I) communication technology on the other hand [24]. Due to the limited computing capabilities, vehicles can outsource the in-car applications to S for execution, with the aim to alleviate the high demand of computing resources. We assume that the offloaded applications belong to a set of K applications, indexed by $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$, and each application a_k can be described as $a_k = (d_k, s_k)$ where d_k denotes the average task-input size (e.g., the processing codes and user related

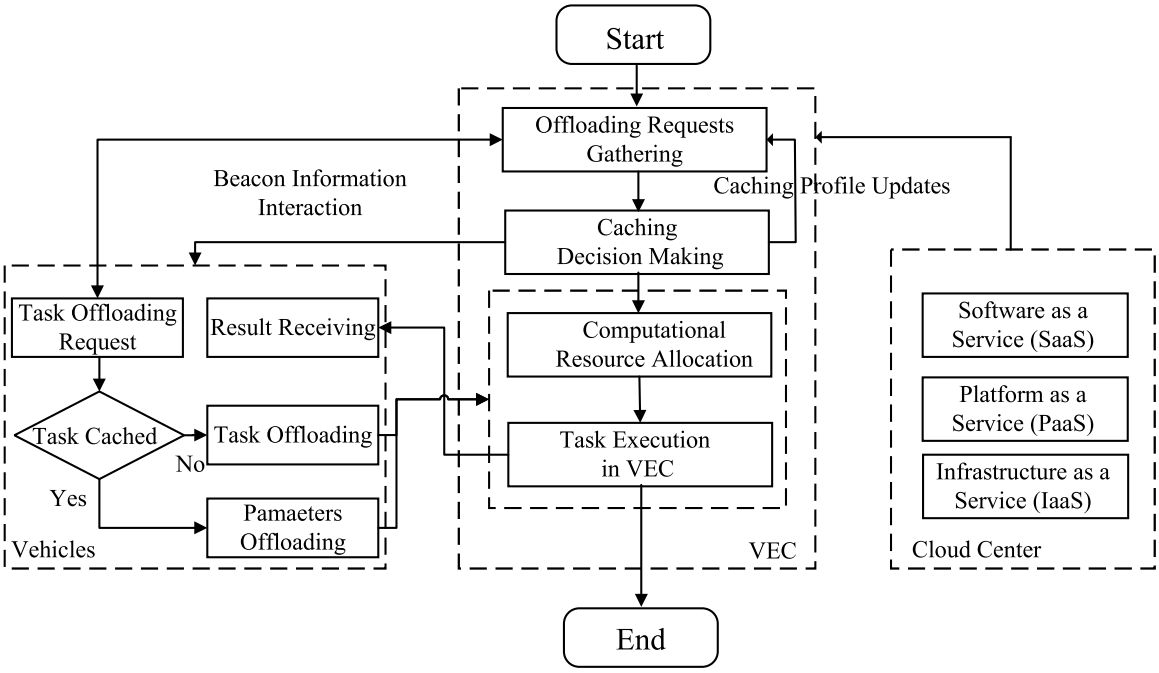


Fig. 2. Sketch of caching-assisted service outsourcing in VEC.

parameters), which need to be offloaded over the wireless channels, and s_k denotes the average number of CPU cycles for accomplishing a_k.

We consider a caching-assisted service outsourcing in VEC in this article, and expect that the response time can be drastically reduced by caching those most frequently requested services/applications at the edge server. For instance, when an application a_k that a vehicle wants to offload is already cached in S, the vehicle does not need to transmit d_k any longer, but to wait for the result to be returned from S. It usually takes a long-term process to evaluate the overall performance of the caching-enabled VEC system. To this end, time is divided into a set of discrete time slots, indexed by $\{0, 1, \ldots, T-1\}$ and each time slot has a duration π. The service caching strategy can be updated within each time slot. Note that we use the three words "applications," "services," and "tasks" exchangeably hereafter.

Owing to the stochastic nature of vehicles, we assume that the arrival of the application offloading requests for a_k in time slot t follows a Poisson process with arrival rate λ_k^t [25]. We define $\delta_k^t \in \{0, 1\}$ as a binary variable to denote whether a_k is cached at S in the tth time slot. If a_k is cached at S, $\delta_k^t = 1$, and 0, otherwise. Thus, the caching decision profile in time slot t can be expressed as $\boldsymbol{\delta}^t = (\delta_1^t, \delta_2^t, \ldots, \delta_K^t)$. We have $\sum_{k=1}^{K} \delta_k^t d_k \leq \mathcal{C}\ \forall t (\in [0, T-1])$ where $\mathcal{C}$ denotes the cache size of S.

Specifically, Fig. 2 shows the sketch of caching-assisted service outsourcing in VEC. For the vehicles with the offloading requests in the vicinity, they can disseminate such requests to VEC (RSU) together with the beacon information. RSU gathers the offloading requests and decides which services should be cached according to the evaluation metric (illustrated later). The caching profile determined by RSU is then sent back to the vehicles. Based on the caching decision, vehicles will perform different offloading operations. For example, if the requested service has been cached in VEC, the vehicle only needs to offload the context information (e.g., the user related parameters). In this case, the time taken to offload the context information is negligible compared to the task itself. On the other hand, if the requested service is not cached in VEC, the vehicle will offload the task to RSU for the execution. Our work in this article focuses on the caching decision making after the information on the requested services is gathered. It shall be noted that at the beginning of each time slot, the caching decision profile can be updated based on the beacon information disseminated from vehicles. For convenient discussion, some key notations to be used are shown in Table I.

A. Service Caching Model

Intuitively, the more the number of services that are cached, the more the response time can be reduced. However, due to the limited cache size of S, it is impractical to cache all the services at the same time. On another hand, from the viewpoint of response latency reduction, it is better for the edge server to maintain the virtual environment for a while after a service is cached. However, the corresponding energy consumptions at S will substantially increase as the number of cached services increases. Therefore, we should judiciously determine which services to be cached at S, for the purpose of response latency reduction while considering the energy consumption.

If application a_k is not cached at S in time slot t, vehicles should offload it to S and wait for the execution result. In this case, the average response time $l_{k,rl}^{t,\mathrm{no}}$ consists of the transmission response $l_{k,\mathrm{trs}}^{t,\mathrm{no}}$ and execution response $l_{k,\mathrm{exe}}^{t,\mathrm{no}}$, each of

TABLE I
NOTATIONS

Notation	Description
K	The number of vehicular applications
S	The edge server deployed at RSU
$\mathcal{A}$	The set of applications to be deployed
d_k	The average size of input data
s_k	The average number of CPU cycles needed for a_k
T	The total number of time slots
π	The duration for each time slot
$\mathcal{C}$	The cache size of S
β_k^t	The number of channels assigned to a_k
r_k^t	The average transmission rate over the channel
λ_k^t	The arrival rate for a_k in time slot t
f_e^t	The average processing frequency of S in time slot t
κ	The effective switched capacitance coefficient
ε	The number of cycles for one task-input bit performing
γ_k	The static power consumption at S for a_k
E_t^{max}	The energy consumption constraint for time slot t
L_t^{max}	The response time constraint for time slot t
Q	The global energy constraint across different slots

which cannot be neglected. However, the time taken to return the result from S can be omitted due to the negligible result size [26]. We need to calculate $l_{k,\text{trs}}^{t,\text{no}}$ and $l_{k,\text{exe}}^{t,\text{no}}$, respectively. The average transmission delay $d_{k,\text{trs}}^{t,\text{no}}$ for a_k over the wireless channel in slot t can be expressed as

$$d_{k,\text{trs}}^{t,\text{no}} = \frac{d_k}{\beta_k^t r_k^t} \tag{1}$$

where β_k^t denotes the number of channels assigned to a_k and r_k^t denotes the average transmission rate over the channel. To simplify the discussion, we assume that r_k^t is a constant, which can be obtained based on the historical statistics. The arrival of tasks at the transmission channel also follows a Poisson process based on the aforementioned assumption. The arrival rate at the transmission queue is λ_k^t and the service rate is $\pi\beta_k^t r_k^t/d_k$. Based on the $M/M/1$ queueing model, the average queuing delay in the transmission channel can be calculated as [27]

$$d_{k,q}^{t,\text{no}} = \frac{\lambda_k^t d_k^2}{\pi\beta_k^t r_k^t\left(\pi\beta_k^t r_k^t - \lambda_k^t d_k\right)}. \tag{2}$$

Since the average transmission response consists of the average queuing delay in the transmission channel and the following average transmission delay, we have:

$$l_{k,\text{trs}}^{t,\text{no}} = d_{k,q}^{t,\text{no}} + d_{k,\text{trs}}^{t,\text{no}}. \tag{3}$$

The execution delay of a_k at S can be calculated as

$$d_{k,\text{exe}}^{t,\text{no}} = d_{k,\text{init}}^{t,\text{no}} + \frac{s_k}{f_e^t} \tag{4}$$

where f_e^t is the average processing frequency of S in time slot t and $d_{k,\text{init}}^{t,\text{no}}$ is the time taken to initialize the VM for a_k in time slot t.

To obtain the queueing delay of a_k, information about other applications $\mathcal{A}\backslash\{a_k\}$ in time slot t is also required. Based on the queueing model, the arrival rate of all the services $\tilde{\lambda}^t = \sum_{k=1}^K \lambda_k^t$ at the task queue of S also follows a Poisson process. Thus, the average execution time (i.e., service time) for a service is $\sum_{k=1}^K \lambda_k^t s_k/\pi f_e^t\tilde{\lambda}^t$. Thus, the service rate for each VM is $\pi f_e^t\tilde{\lambda}^t/\sum_{k=1}^K \lambda_k^t s_k$. The number of VMs in S is assumed to be m. According to the $M/M/s$ queueing theory, the average queueing delay for a_k is given as

$$d_{k,aq}^{t,\text{no}} = \frac{m\rho}{\tilde{\lambda}^t} + \frac{m^m\rho^{m+1}P_0}{\tilde{\lambda}^t m!} \tag{5}$$

where $\rho = \tilde{\lambda}^t\sum_{k=1}^K \lambda_k^t s_k/(m\pi f_e^t\tilde{\lambda}^t)$ and $P_0 = 1/(\sum_{k=0}^{m-1}(m\rho)^k/k! + (m\rho)^m/(m!(1-\rho)))$. Thus, the average execution response time $l_{k,\text{exe}}^{t,\text{no}}$ can be expressed as

$$l_{k,\text{exe}}^{t,\text{no}} = d_{k,aq}^{t,\text{no}} + d_{k,\text{exe}}^{t,\text{no}}. \tag{6}$$

Furthermore, the average energy consumption for a_k at S can be given as

$$e_k^{t,\text{no}} = \kappa\varepsilon s_k\left(f_e^t\right)^2 \tag{7}$$

where κ and ε represent the effective switched capacitance coefficient and the number of cycles required for one task-input bit performing at S, respectively.

If a_k is cached at S in time slot t, it is unnecessary for vehicles to offload the corresponding tasks to S. In this case, the average response time of a_k denoted by $l_{k,rl}^{t,c}$ is directly equal to the execution response time $l_{k,\text{exe}}^{t,c}$. In most of the existing works, $l_{k,\text{exe}}^{t,c} = 0$ holds, as long as the service is cached. The assumption behind it is that the computational result is applicable to any offloading request irrespective of context information and user related parameters. As a contrast, we allow for the diversity of context information and user parameters. As a result, $l_{k,\text{exe}}^{t,c}$ needs to be recalculated even if the corresponding service is cached. The execution delay of a_k in time slot t is: $d_{k,\text{exe}}^{t,c} = s_k/f_e^t$. Since the running state of the virtual environment is maintained when the service is cached, the initialization time can be omitted, which is different from $d_{k,\text{exe}}^{t,\text{no}}$. The average queueing delay for a_k, i.e., $d_{k,aq}^{t,c}$, is the same as $d_{k,aq}^{t,\text{no}}$. Thus, when the corresponding service is cached, the average execution response consisting of the average queueing delay and average execution delay can be expressed as

$$l_{k,\text{exe}}^{t,c} = d_{k,aq}^{t,c} + d_{k,\text{exe}}^{t,c}. \tag{8}$$

The corresponding energy consumption can be calculated as

$$e_k^{t,c} = \gamma_k + \kappa\varepsilon s_k\left(f_e^t\right)^2 \tag{9}$$

where γ_k is the static power consumption caused by the maintenance of the virtual environment for a_k regardless of the workloads. Therefore, we have the expected energy consumption E_t in time slot t as follows:

$$E_t\left(\boldsymbol{\delta}^t\right) = \sum_{k=1}^K\left(1-\delta_k^t\right)e_k^{t,\text{no}} + \delta_k^t e_k^{t,c}. \tag{10}$$

B. Problem Formulation

Our goal in this article is to minimize the average response time over a long time-slotted horizon while keeping the energy

consumption at S below the given threshold. Accordingly, the optimization problem is formulated as follows:

$$(P1)\ \min_{\boldsymbol{\delta}^t\ \forall t}\ \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K}\Big[\big(1-\delta_k^t\big)l_{k,rl}^{t,\mathrm{no}}+\delta_k^t l_{k,rl}^{t,c}\Big] \tag{11}$$

$$\text{s.t.}\quad \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}E_t\big(\boldsymbol{\delta}^t\big)\le Q \tag{12}$$

$$\sum_{k=1}^{K}\delta_k^t d_k\le\mathcal{C}\quad\forall t\in[1,T] \tag{13}$$

$$E_t\big(\boldsymbol{\delta}^t\big)\le E_t^{\max}\quad\forall t\in[1,T] \tag{14}$$

$$\sum_{k=1}^{K}\Big[\big(1-\delta_k^t\big)l_{k,rl}^{t,\mathrm{no}}+\delta_k^t l_{k,rl}^{t,c}\Big]\le L_t^{\max}\quad\forall t\in[1,T] \tag{15}$$

where $L_t^{\max}$ and $E_t^{\max}$ denote the maximum response time and energy consumption allowed for time slot t, respectively. Specifically, conditions (14) and (15) place $L_t^{\max}$ and $E_t^{\max}$ on the per-slot response time and energy consumptions, respectively. Condition (12) represents that the average energy consumption over the accumulated time slots should not go beyond the energy consumption constraint Q. Constraint (13) ensures that the amount of services to be cached should not exceed the cache size of S.

Challenges: Exhaustive search over the potential solution space is prohibitively costly, since it takes the exponential time to determine the best caching profile for each time slot. Furthermore, to optimally solve problem $P1$ requires the future information (e.g., service distributions in all time slots). Based on these information, the optimal solution can be obtained in an offline way. However, it is highly difficult to predict such information beforehand in reality. Accordingly, these challenges necessitate an online approach, which can efficiently make service caching decisions without the future information.

IV. Response Time Minimization Strategy Considering Energy Consumption in Caching-Assisted VEC

A. Lyapunov-Based Online Caching Decision Algorithm

Directly solving problem $P1$ is challenged by not only the prediction on the future information but also the entire energy consumption constraint across the time slots. Accordingly, the Lyapunov optimization framework is adopted in this article to tackle such challenges. We construct a dynamic energy migration queue to assist the caching decision making while satisfying the long-term energy constraint. Let $q(0)=0$, and this queue can be recursively defined as

$$q(t+1)=\max\big[q(t)-Q,0\big]+E_t\big(\boldsymbol{\delta}^t\big) \tag{16}$$

where $q(t)$ is the queue backlog in time slot t that reflects the deviation of current energy consumption from the energy constraint Q. Hence, the larger the value of $q(t)$, the sharper the deviation.

Lemma 1: Given the queue backlog $q(t)$, the following inequality holds:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[E_t\big(\boldsymbol{\delta}^t\big)-Q\big]\le\frac{1}{T}\mathbb{E}[q(T)].$$

Proof: We can briefly prove it as follows. If $q(t)\ge Q$, according to the definition of $q(t)$, we have $q(t+1)=\max[q(t)-Q,0]+E_t(\boldsymbol{\delta}^t)=q(t)-Q+E_t(\boldsymbol{\delta}^t)$; thus, $E_t(\boldsymbol{\delta}^t)-Q=q(t+1)-q(t)$. If $q(t)<Q$, we have $q(t+1)=\max[q(t)-Q,0]+E_t(\boldsymbol{\delta}^t)=E_t(\boldsymbol{\delta}^t)$ and $E_t(\boldsymbol{\delta}^t)-Q<q(t+1)-q(t)$. Accordingly, $E_t(\boldsymbol{\delta}^t)-Q\le q(t+1)-q(t)\ \forall t\in\{0,\ldots,T-1\}$. Taking the expectation of this inequality and further summarizing it over $\forall t\in\{0,\ldots,T-1\}$, we have

$$\begin{aligned}\sum_{t=0}^{T-1}\mathbb{E}\big[E_t\big(\boldsymbol{\delta}^t\big)-Q\big]&\le\sum_{t=0}^{T-1}\mathbb{E}\big[q(t+1)-q(t)\big]\\&=\mathbb{E}[q(T)].\end{aligned}$$

Thus

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[E_t\big(\boldsymbol{\delta}^t\big)-Q\big]\le\frac{1}{T}\mathbb{E}[q(T)].$$

■

The Lyapunov function can be defined as $L(q(t))=(1/2)q^2(t)$. To ensure the strong stability of this migration queue, the increment between two consecutive states should be as small as possible. To this end, we define the *Lyapunov drift* as $\Delta(q(t))\triangleq\mathbb{E}[L(q(t+1))-L(q(t))|q(t)]$. We can determine the upper bound of $\Delta(q(t))$, based on the Lemma from [28], given as follows.

Lemma 2: Assuming A, B, C, and m are non-negative real numbers and $C=\max\{B-m,0\}+A$, then $C^2\le A^2+B^2+m^2-2B(m-A)$.

Based on this lemma, we have

$$\begin{aligned}\Delta(q(t))&=\mathbb{E}\big[L(q(t+1))-L(q(t))|q(t)\big]\\&=\frac{1}{2}\mathbb{E}\Big[\big[\max\big[q(t)-Q,0\big]+E_t\big(\boldsymbol{\delta}^t\big)\big]^2-q^2(t)|q(t)\Big]\\&\le\frac{1}{2}\mathbb{E}\Big[Q^2+E_t^2\big(\boldsymbol{\delta}^t\big)+2q(t)\big(E_t\big(\boldsymbol{\delta}^t\big)-Q\big)|q(t)\Big]\\&=\frac{1}{2}Q^2+\mathbb{E}\left[\frac{E_t^2\big(\boldsymbol{\delta}^t\big)}{2}-Qq(t)+q(t)E_t\big(\boldsymbol{\delta}^t\big)|q(t)\right]\\&=D-Qq(t)+\mathbb{E}\big[q(t)E_t\big(\boldsymbol{\delta}^t\big)|q(t)\big]\end{aligned} \tag{17}$$

where $D=(Q^2/2)+\mathbb{E}[([E_t^2(\boldsymbol{\delta}^t)]/2)|q(t)]\le(Q^2/2)+\mathbb{E}[([(E_t^{\max})^2]/2)|q(t)]=(Q^2/2)+([(E_t^{\max})^2]/2)\triangleq B$. As a result, $\Delta(q(t))\le B-Qq(t)+\mathbb{E}[q(t)E_t(\boldsymbol{\delta}^t)|q(t)]$. It is easily observed that the upper bound of $\Delta(q(t))$ does not need the future information such as $q(t+1)$ at time slot t. Therefore, we can convert the time slot spanned energy constraint into per-slot ones by means of the Lyapunov optimization technology. Specifically, the *drift-plus-penalty* term in each time slot is given as

$$\begin{aligned}&\Delta(q(t))+V\mathbb{E}\big[L_t\big(\boldsymbol{\delta}^t\big)|q(t)\big]\\&\quad\le B-Qq(t)+\mathbb{E}\big[q(t)E_t\big(\boldsymbol{\delta}^t\big)|q(t)\big]+V\mathbb{E}\big[L_t\big(\boldsymbol{\delta}^t\big)|q(t)\big]\\&\quad=B-Qq(t)+\mathbb{E}\big[q(t)E_t\big(\boldsymbol{\delta}^t\big)+VL_t\big(\boldsymbol{\delta}^t\big)|q(t)\big]\end{aligned} \tag{18}$$

where $L_t(\boldsymbol{\delta}^t) \triangleq \sum_{k=1}^{K}[(1-\delta_k^t)l_{k,rl}^{t,\mathrm{no}}+\delta_k^t l_{k,rl}^{t,c}]$ is the response time at time slot t given the caching decision $\boldsymbol{\delta}^t$. $V(>0)$ is a control parameter to adjust the tradeoff between energy consumption control and response time minimization. Now, our attention has shifted from $P1$ to the minimization of the supremum of the *drift-plus-penalty* term, i.e., the right-hand side of (18). Namely, the caching decision $\boldsymbol{\delta}^t$ can be determined for time slot t by solving the following problem $P2$:

$$(P2)\ \min_{\forall t,\boldsymbol{\delta}^t}\ \left\{q(t)E_t\left(\boldsymbol{\delta}^t\right)+VL_t\left(\boldsymbol{\delta}^t\right)\right\} \tag{19}$$

$$\text{s.t. (13)}-\text{(15)}. \tag{20}$$

It is worth mentioning that the constraint (12) in $P1$ has been incorporated into problem $P2$ itself by means of $\Delta q(t)$.

Theorem 1: Based on the drift-plus-penalty term, the caching decisions obtained by solving $P2$ over time slots $t \in \{0,\ldots,T-1\}$ are approximately optimal caching decisions w.r.t. $P1$.

Proof: If there is a solution to $P1$, then there exists a caching decision $\boldsymbol{\delta}^{t*}$ and l^*, while satisfying: 1) $E_t(\boldsymbol{\delta}^{t*}) \le Q$ and 2) $l^* = \min_{\boldsymbol{\delta}^t\ \forall t}\ \lim_{T\to\infty}(1/T)\sum_{t=1}^{T}\sum_{k=1}^{K}L_t(\boldsymbol{\delta}^t)$, based on [29]. Therefore, we have

$$\begin{aligned}
&\Delta(q(t))+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\\
&\quad\le B-Qq(t)+\mathbb{E}\left[q(t)E_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\\
&\quad= B+\mathbb{E}\left[q(t)\left(E_t\left(\boldsymbol{\delta}^t\right)-Q\right)|q(t)\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\\
&\quad= B+q(t)\mathbb{E}\left[\left(E_t\left(\boldsymbol{\delta}^t\right)-Q\right)|q(t)\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\\
&\quad\overset{\ddagger}{\le} B+Vl^*.
\end{aligned}$$

Since $\Delta(q(t))+V\mathbb{E}[L_t(\boldsymbol{\delta}^t)|q(t)] \le B+q(t)\mathbb{E}[(E_t(\boldsymbol{\delta}^t)-Q)|q(t)]+V\mathbb{E}[L_t(\boldsymbol{\delta}^t)|q(t)]$, for $\forall t\in\{0,\ldots,T-1\}$ and valid caching decision $\boldsymbol{\delta}^t$, the inequality still holds when substituting $\boldsymbol{\delta}^t$ by $\boldsymbol{\delta}^{t*}$, i.e., $B+q(t)\mathbb{E}[(E_t(\boldsymbol{\delta}^{t*})-Q)|q(t)]+V\mathbb{E}[L_t(\boldsymbol{\delta}^{t*})|q(t)]$. Owing to $E_t(\boldsymbol{\delta}^{t*})-Q\le 0$, the inequality ($\ddagger$) holds.

Then, we calculate the expectation of the above inequality and then perform a sum of the expectation over the time slots $t\in\{0,\ldots,T-1\}$, namely

$$\begin{aligned}
&\sum_{t=0}^{T-1}\mathbb{E}\left[\Delta(q(t))+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\right]\\
&\quad=\sum_{t=0}^{T-1}\mathbb{E}\left[\mathbb{E}\left[L(q(t+1))-L(q(t))|q(t)\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)|q(t)\right]\right]\\
&\quad=\sum_{t=0}^{T-1}\mathbb{E}\left[L(q(t+1))-L(q(t))\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)\right]\\
&\quad=\mathbb{E}\left[L(q(T))-L(q(0))\right]+\sum_{t=0}^{T-1}V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)\right]\\
&\quad=\mathbb{E}[L(q(T))]+\sum_{t=0}^{T-1}V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^t\right)\right]\le\sum_{t=0}^{T-1}\mathbb{E}\left[B+Vl^*\right]\\
&\quad=\left(B+Vl^*\right)*T.
\end{aligned}$$

Since $\mathbb{E}[L(q(T))]\ge 0$, we have

$$\sum_{t=0}^{T-1}V\mathbb{E}[L_t(\boldsymbol{\delta}^t)]\le(B+Vl^*)*T.$$

Then

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[L_t(\boldsymbol{\delta}^t)]\le l^*+\frac{B}{V}.$$

Namely

$$\frac{1}{T}\sum_{t=0}^{T-1}L_t(\boldsymbol{\delta}^t)\le l^*+\frac{B}{V}.$$

This is because $L_t(\boldsymbol{\delta}^t)$ is the expected response time, and according to the law of iterated expectations, $\mathbb{E}[L_t(\boldsymbol{\delta}^t)] = L_t(\boldsymbol{\delta}^t)$. Thus, the caching decision obtained by solving $P2$ can be infinitely close to the solution of $P1$ by adjusting the variable V. ■

Remark: Let $\mathcal{G}(V)$ represent the optimality gap, which denotes the difference between l^* and the solution of $P2$. The supremum of $P2$ is l^*+B/V, so $\mathcal{G}(V)=B/V$ and $O(\mathcal{G}(V))=O(l^*+B/V)=O(1/V)$. As a consequence, if problem $P2$ can be solved within each time slot, $\mathcal{G}(V)$ can be bounded by $O(1/V)$.

Theorem 2: The caching decisions obtained by solving problem $P2$ over time slot $t\in\{0,\ldots,T-1\}$ make inequality (12) always true.

Proof: Let $\boldsymbol{\delta}^{t*}$ denote the caching decision obtained by solving problem $P2$, i.e., the right-hand side of drift-plus-penalty inequality (18) in time slot t. Thus, we have

$$\begin{aligned}
&\Delta(q(t))+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^{t*}\right)|q(t)\right]\\
&\quad\le B-Qq(t)+\mathbb{E}\left[q(t)E_t\left(\boldsymbol{\delta}^{t\ddagger}\right)|q(t)\right]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^{t\ddagger}\right)|q(t)\right]\\
&\quad\le B-Qq(t)+VL_t^{\ddagger}+q(t)(Q-\epsilon)\\
&\quad= B-\epsilon q(t)+VL_t^{\ddagger}
\end{aligned}$$

where $\boldsymbol{\delta}^{t\ddagger}$ is any other valid caching decision in time slot t except $\boldsymbol{\delta}^{t*}$. $L_t^{\ddagger}$ is the corresponding response time with caching decision equal to $\boldsymbol{\delta}^{t\ddagger}$. Since $\mathbb{E}[E_t(\boldsymbol{\delta}^{t*})]=E_t(\boldsymbol{\delta}^{t*})\le Q$ [15], there exists a small enough $\epsilon>0$, satisfying $E_t(\boldsymbol{\delta}^{t*})-Q\le-\epsilon$, namely, $E_t(\boldsymbol{\delta}^{t*})\le Q-\epsilon$.

By taking expectations of the above inequality, we have: $\mathbb{E}\{\Delta(q(t))+V\mathbb{E}[L_t(\boldsymbol{\delta}^{t*})|q(t)]\}\le B-\epsilon\mathbb{E}[q(t)]+VL_t^{\ddagger}$, namely

$$\begin{aligned}
&\mathbb{E}\left[L(q(t+1))\right]-\mathbb{E}[L(q(t))]+V\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^{t*}\right)\right]\\
&\quad\le B-\epsilon\mathbb{E}[q(t)]+VL_t^{\ddagger}.
\end{aligned}$$

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

$$\begin{aligned}
&\mathbb{E}\left[L(q(T))\right]-\mathbb{E}\left[L(q(0))\right]+V\sum_{t=0}^{T-1}\mathbb{E}\left[L_t\left(\boldsymbol{\delta}^{t*}\right)\right]\\
&\quad\le BT+VL_t^{\ddagger}T-\epsilon\mathbb{E}[q(t)]T\\
&\quad\le BT+VL_t^{\ddagger}T=B'T
\end{aligned}$$

where $B'=B+VL_t^{\ddagger}$. Since $V\sum_{t=0}^{T-1}\mathbb{E}[L_t(\boldsymbol{\delta}^{t*})]\ge 0$, thus $\mathbb{E}[L(q(T))]-\mathbb{E}[L(q(0))]\le BT+VL_t^{\ddagger}T$. Substituting $L(q(T))$ by $L(q(T))=(1/2)q^2(T)$, we have

$$\frac{1}{2}\mathbb{E}\left[q^2(T)\right]\le B'T+\frac{1}{2}\mathbb{E}\left[q^2(0)\right]=B'T.$$

Algorithm 1: LOCD

Input: $q(0)$, Q, $\mathcal{C}$, $\mathcal{A}$, V, T
Output: Optimum solution to $P1$

1 $L = 0$;
2 **for** $t = 0$ to $T-1$ **do**
3 Observe and record $\boldsymbol{\lambda}^t$, $\boldsymbol{r}^t$;
4 Set L_t^{max} and E_t^{max};
5 Obtain $\boldsymbol{\delta}^{t*}$ by solving problem $P2$;
6 $L = L + L_t(\boldsymbol{\delta}^{t*})$;
7 Calculate $E_t(\boldsymbol{\delta}^t)$ based on Eq. 10;
8 $q(t+1) = \max[q(t) - Q, 0] + E_t(\boldsymbol{\delta}^t)$;
9 **end**
10 $avg = L/T$;
11 return avg;

According to the Cauchy inequality, i.e., $(\sum_{i=1}^n x_i y_i)^2 \le (\sum_{i=1}^n x_i^2)(\sum_{i=1}^n y_i^2)$, we have

$$(\mathbb{E}[q(T)])^2 \le \mathbb{E}\Big[q^2(T)\Big] \le 2B'T.$$

Thus, $\mathbb{E}[q(T)] \le \sqrt{2B'T}$, so

$$\frac{1}{T}\mathbb{E}[q(T)] \le \sqrt{\frac{2B'}{T}}.$$

Based on Lemma 1, we have

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[E_t(\boldsymbol{\delta}^t) - Q\big] \le \frac{1}{T}\mathbb{E}[q(T)] \le \sqrt{\frac{2B'}{T}}.$$

Therefore

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[E_t(\boldsymbol{\delta}^t) - Q\big] \le \lim_{T\to\infty}\sqrt{\frac{2B'}{T}} = 0.$$

Namely,

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\big[E_t(\boldsymbol{\delta}^t) - Q\big] \le 0.$$

As a consequence, we can see that: 1) the queue backlog is mean rate stable and 2) the caching decision obtained by solving $P2$ can solve $P1$ without constraint violation. ■

Based on the above descriptions, we have proposed a Lyapunov-based online caching decision (LOCD) algorithm to obtain approximately optimum solution to problem $P1$, as shown in Algorithm 1. Specifically, LOCD seeks the optimal solution along each time slot. Within each time slot, given per-slot constraints on the expected response time and energy consumption (i.e., $L_t^{\max}$ and $E_t^{\max}$), respectively, we strive to achieve approximately optimum solution by solving problem $P2$ (lines 3–5). Then, the per-slot best caching decision can be obtained, i.e., $\boldsymbol{\delta}^{t*}$. LOCD calculates the optimal response time based on $\boldsymbol{\delta}^{t*}$ and then the per-slot response time is accumulated (lines 7 and 8). Finally, the mean value of the objective function (i.e., avg $= L/T$) can be retrieved.

B. Algorithm Design for Solving $P2$

Let $\boldsymbol{\delta}^{t*} = \arg\min_{\boldsymbol{\delta}^t} \forall_t\{q(t)E_t(\boldsymbol{\delta}^t) + VL_t(\boldsymbol{\delta}^t)\}$ while satisfying the constraints (13)–(15). Then, $\boldsymbol{\delta}^{t*}$ can be viewed as the best caching decision in time slot t in terms of $L_t(\boldsymbol{\delta}^t)$. Given $q(t)$ and V, the problem $P2$ is actually a 0-1 knapsack problem. Owing to its well-known NP-hardness, exhaustive search takes exponential time and thus, makes itself infeasible in reality. Evolutionary algorithms, such as GA and PSO [14], [20] can be adopted to solve it. However, these iteration-based methods are usually time consuming and thus, impracticable in time-sensitive scenarios. Accordingly, a heuristic algorithm suitable for the time-sensitive scenario is required to solve $P2$. Let $\mathcal{F} = q(t)E_t(\boldsymbol{\delta}^t) + VL_t(\boldsymbol{\delta}^t)$ and by substituting $E_t(\boldsymbol{\delta}^t)$ and $L_t(\boldsymbol{\delta}^t)$, respectively, we have

$$\begin{aligned}\mathcal{F}\big(\boldsymbol{\delta}^t\big) &= q(t)E_t\big(\boldsymbol{\delta}^t\big) + VL_t\big(\boldsymbol{\delta}^t\big)\\ &= \sum_{k=1}^{K}\Big[q(t)e_k^{t,\mathrm{no}} + Vl_{k,rl}^{t,\mathrm{no}}\Big]\\ &\quad - \sum_{k=1}^{K}\delta_k^t\Big[q(t)\gamma_k + V\Big(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c}\Big)\Big]\\ &\triangleq \mathcal{P} - \mathcal{G}\end{aligned} \tag{21}$$

where $\mathcal{P} = \sum_{k=1}^{K}[q(t)e_k^{t,\mathrm{no}} + Vl_{k,rl}^{t,\mathrm{no}}]$, independent of the caching decision $\boldsymbol{\delta}^t$, can be determined within each time slot, and $\mathcal{G} = \sum_{k=1}^{K}\delta_k^t[q(t)\gamma_k + V(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c})]$ depends upon $\boldsymbol{\delta}^t$. To minimize $\mathcal{F}$ means to maximinze $\mathcal{G}$ in essence, since $\mathcal{P}$ is unchangeable given the time slot t.

To meet the strict response time requirement, a greedy algorithm is proposed to minimize $\mathcal{F}$. In particular, a greedy heuristic is given in what follows. Let $g_k^t = q(t)\gamma_k + V(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c})$, and the applications in $\mathcal{A}$ are sorted in the descending order of g_k^t/d_k. The corresponding greedy algorithm is shown in Algorithm 2. A list $\mathcal{L}$ is used to store the sorted applications. S_0 and S_1 store the indexes of tasks that are cached and not cached, respectively. Therefore, the caching decision profile $\boldsymbol{\delta}^t$ can be easily constructed based on S_0 and S_1. Initially, $S_1 = \emptyset$ and $S_0 = \{0, 1, \ldots, K-1\}$, respectively. We check application a_k in $\mathcal{L}$ one by one. If the sum of input data of all already cached applications plus current a_k does not exceed the constraint $\mathcal{C}$, we tend to cache a_k. However, the per-slot energy and latency constraints are examined, respectively, before updating the decision caching profile (lines 12–15). In the meanwhile, the accumulated energy consumption and response time are obtained, respectively, (lines 10 and 11). If the sum of input data of all already cached applications plus current a_k exceeds the constraint $\mathcal{C}$, the algorithm checks the application after a_k in $\mathcal{L}$. In the meanwhile, the accumulated energy consumption and response time are also calculated. Based on (21), the minimal $\mathcal{F}$ can be retrieved. Finally, S_0 and S_1 are returned.

Remark: The time of GASP is mainly taken to sort the applications in $\mathcal{A}$. Various sort algorithms can be utilized to achieve this goal with the time complexity of $O(K\log K)$, where K is the number of applications. In addition, the time taken to check applications in $\mathcal{A}$ is $O(K)$. Thus, the time complexity of GASP is $O(K\log K) + O(K) = O(K\log K)$, which is

Algorithm 2: Greedy Algorithm for Solving P2 (GASP)

Input: $q(t)$, $e_k^{t,no}$, $l_{k,rl}^{t,c}$, $l_{k,rl}^{t,no}$, $\mathcal{C}$, $\mathcal{A}$, V,γ_k
Output: S_0 and S_1
$S_0 = \{0, 1, ..., K-1\}$;
$S_1 = \emptyset$;
$g_k^t = q(t)\gamma_k + V(l_{k,rl}^{t,no} - l_{k,rl}^{t,c})$, for $\forall k \in [1, K]$;
$\mathcal{L}$ stores the applications in descending order of g_k^t/d_k;
$s = 0, L = 0, E = 0$;
for each a_k in $\mathcal{L}$ **do**
  $s = s + d_k$;
  if $s \le \mathcal{C}$ **then**
    Calculate $l_{k,rl}^{t,c}$ and $e_k^{t,c}$;
    $L = L + l_{k,rl}^{t,c}$;
    $E = E + e_k^{t,c}$;
    if $L \le L_t^{max}$ *and* $E \le E_t^{max}$ **then**
      $\delta_k^t = 1$;
      $S_1 = S_1 \cup \{k\}$;
      $S_0 = S_0 \backslash \{k\}$;
    else
      $L = L - l_{k,rl}^{t,c}$;
      $E = E - e_k^{t,c}$;
    end
  else
    $s = s - d_k$;
    Calculate $l_{k,rl}^{t,no}$ and $e_k^{t,no}$;
    $L = L + l_{k,rl}^{t,no}$;
    $E = E + e_k^{t,no}$;
  end
end
$\mathcal{F}_{min} = q(t)E + VL$;
return S_0 and S_1;

much better than the evolutionary algorithms. GASP endeavors to obtain the approximate optimal solution according to the greedy heuristic. However, similar to the general 0-1 knapsack problem solved by greedy algorithms, the drawback of GASP is that the vacant part of knapsack can depreciate the value of knapsack if the knapsack is not fully occupied. Therefore, GASP needs to be enhanced for the performance improvement.

C. Enhanced Algorithm for Solving Problem P2

To narrow down the optimality gap between the approximate optimal value and true optimal value, in this section, we apply two other greedy rules to the approximate solution obtained by GASP.

Definition 1: 1-D increment $\Delta_k(\boldsymbol{\delta}^t)$ is defined as $\Delta_k(\boldsymbol{\delta}^t) = \mathcal{G}(\delta_1^t, \ldots, 1_{(k)}, \ldots, \delta_K^t) - \mathcal{G}(\delta_1^t, \ldots, 0_{(k)}, \ldots, \delta_K^t) = q(t)(e_k^{t,\mathrm{no}} - e_k^{t,c}) + V(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c}) = q(t)\gamma_k + V(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c})$.

$\Delta_k(\boldsymbol{\delta}^t)$ can be used to evaluate the optimum increment to the objective function $\mathcal{G}$ by unilaterally changing δ_k^t from 0 to 1 while keeping other caching decisions $\{\delta_i^t | 1 \le i \le K, i \ne k\}$ unchanged.

Definition 2: 2-D increment $\Delta_{kl}(\boldsymbol{\delta}^t)$ is defined as $\Delta_{kl}(\boldsymbol{\delta}^t) = \mathcal{G}(\delta_1^t, \ldots, 1_{(k)}, \ldots, 0_{(l)}, \ldots, \delta_K^t) - \mathcal{G}(\delta_1^t, \ldots, 0_{(k)}, \ldots, 1_{(l)}, \ldots, \delta_K^t) = q(t)(\gamma_k - \gamma_l) + V(l_{k,rl}^{t,\mathrm{no}} - l_{k,rl}^{t,c} - l_{l,rl}^{t,\mathrm{no}} + l_{l,rl}^{t,c})$.

Algorithm 3: EnGASP

Input: $\mathcal{F}_{min}$, $\boldsymbol{\delta}^t$, S_0, S_1
Output: $\boldsymbol{\delta}^t$
1 **while** $S_0 \ne \emptyset$ **do**
2 Obtain k^* by solving: $k^* = \arg\max\{\Delta_k(\boldsymbol{\delta}^t) | k \in S_0\}$;
3 **if** $\sum_i \{d_i | i \in S_1\} + d_{k^*} \le \mathcal{C}$ **then**
4 Update $\mathcal{F}_{min}$;
5 $\delta_{k^*}^t = 1$;
6 $S_1 = S_1 \cup \{k^*\}$;
7 $S_0 = S_0 \backslash \{k^*\}$;
8 **else**
9 $S_0 = S_0 \backslash \{k^*\}$;
10 **end**
11 **end**
12 $S_0 = \{1, ..., K\} \backslash S_1$;
13 **while** $S_0 \ne \emptyset$ and $S_1 \ne \emptyset$ **do**
14 Obtain (k^*, l^*) by solving:
  $(k^*, l^*) = \arg\min\{\Delta_{kl}(\boldsymbol{\delta}^t) | k \in S_1, l \in S_0\}$;
15 **if** $\Delta_{k^*l^*}(\boldsymbol{\delta}^t) < 0$ and $\sum_i \{d_i | i \in S_1 \backslash \{k^*\} \cup \{l^*\}\} \le \mathcal{C}$ **then**
16 Update $\mathcal{F}_{min}$;
17 $\delta_{k^*}^t = 0$ and $\delta_{l^*}^t = 1$;
18 $S_1 = S_1 \backslash \{k^*\}$;
19 $S_0 = S_0 \backslash \{l^*\}$;
20 **end**
21 **end**
22 return $\boldsymbol{\delta}^t$;

$\Delta_{kl}(\boldsymbol{\delta}^t)$ can denote the optimum increment to $\mathcal{G}$ by swapping the caching decisions of two services (e.g., service k from S_1 and l from S_0, respectively), with one varying from 0 to 1 and the other from 1 to 0. If $\Delta_{kl}(\boldsymbol{\delta}^t) < 0$, the objective function $\mathcal{F}$ can be further optimized by exchanging the caching decisions of two tasks from S_1 and S_0, respectively.

Accordingly, we can improve the performance of GASP with the aid of $\Delta_k(\boldsymbol{\delta}^t)$ and $\Delta_{kl}(\boldsymbol{\delta}^t)$. To be specific, given a decision profile $\boldsymbol{\delta}^t$ obtained by GASP, i.e., S_0 and S_1, the procedure of enhancing GASP, denoted by enhanced GASP (EnGASP), is shown in Algorithm 3. EnGASP mainly consists of two steps, with one called the filling stage (lines 1–11) and the other called the swapping stage (lines 13–21). The filling stage, targeted at the vacant part of knapsack, strives to fill the knapsack with uncached applications from S_0. The swapping stage on the other hand aims to find a better caching decision profile by swapping two applications from S_1 and S_0, respectively. For example, $\Delta_{k^*l^*}(\boldsymbol{\delta}^t)$ smaller than 0 means that $\mathcal{G}$ can be increased by exchanging the caching decisions of applications a_{k^*} and a_{l^*}. In the meanwhile, if the caching size constraint can be satisfied after the exchanging, $\mathcal{F}$ can be further optimized by increasing $\mathcal{G}$.

Remark: It is worth mentioning that EnGASP attempts to enhance GASP rather than replacing it, since the input parameters of EnGASP are the output parameters of GASP. Intuitively, the performance of EnGASP is at least as good as

TABLE II
PARAMETER SETTINGS

Parameter	Value	Parameter	Value
T	[0, 1000]	K	20
d_k	[1, 50]	s_k	[40, 70]
$\mathcal{C}$	200	r_k^t	150
β_k^t	[1, 3]	λ_k^t	[100, 200]
π	400	f_e^t	[1000, 2000]
m	3	$\kappa\varepsilon$	1e-5
Q	1.2	E_t^{max}	20
L_t^{max}	100	V	[300, 1000]
γ_k^t	(0, 0.1)	$d_{k,init}^{t,no}$	(0, 1)

GASP. However, as for the time complexity of EnGASP, we can see that the main two loops (i.e., lines 1 and 13) take $O(K)$ and $O(K)$, respectively. To obtain k^* usually needs to traverse the set S_0, which takes the time of $O(K)$. On the other hand, to obtain (k^*, l^*) usually requires to find the pair of (l, k) by traversing S_0 and S_1, which takes time of $O(|S_1|)$ and $O(|S_0|)$, respectively. Thus, the time to find (k^*, l^*) is $O(|S_0| \cdot |S_1|)$. Since $O(|S_0| \cdot |S_1|) \leq O(((|S_0| + |S_1|)/2)^2) = O(K^2/4) = O(K^2)$, the time complexity of EnGASP can be derived as $O(K) \cdot O(K) + O(K) \cdot O(K^2) = O(K^3)$. Compared to the iteration-based evolutionary algorithms [14], [20], both GASP and EnGASP are of polynomial time. Therefore, both are really appropriate for time-sensitive scenarios in VEC systems. In addition, extensive simulation is still needed to evaluate EnGASP and GASP w.r.t. efficiency and effectiveness.

V. NUMERIC EVALUATION

Extensive experiments have been carried out to evaluate the proposed caching strategy in terms of response time and energy consumption in this section. In what follows, we will report the experimental settings and simulation results, respectively.

A. Experimental Settings

The involved parameters with the corresponding values are shown in Table II. Although the involved parameters in the simulation are given manually, our previous works [14], [20] can bring rich experience to the parameter settings, including the duration for each time slot π, $\kappa\varepsilon$, f_e^t, and so on. For instance, π should be set appropriately such that on the one hand, the edge server S has enough time to make caching decisions, and on the other hand, tasks offloaded to S can be completely accomplished by the end of time slot. The number of time slots varies from 0 to 1000 and the number of applications is set to 20. Within each time slot, these vehicular applications are randomly generated according to d_k and s_k. For the caching-related parameters, the global energy constraint Q across different time slots is set to 1.2. For simplicity, we assume that the maximal per-slot energy consumption constraint is the same for each time slot (i.e., 20) in the simulation. The control parameter for the drift-plus-penalty term V ranges from 300 to 1000.

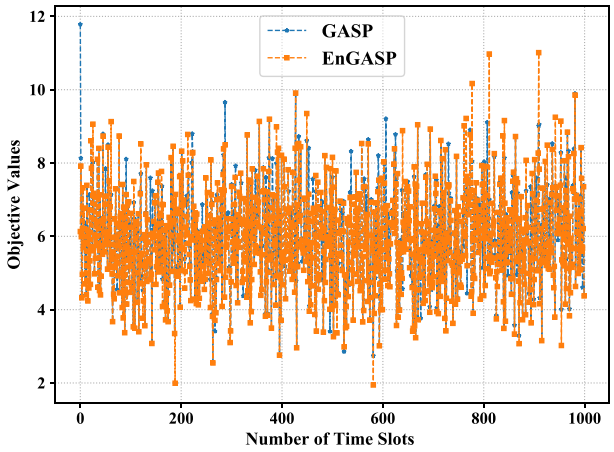


Fig. 3. Performance comparison with different time slots.

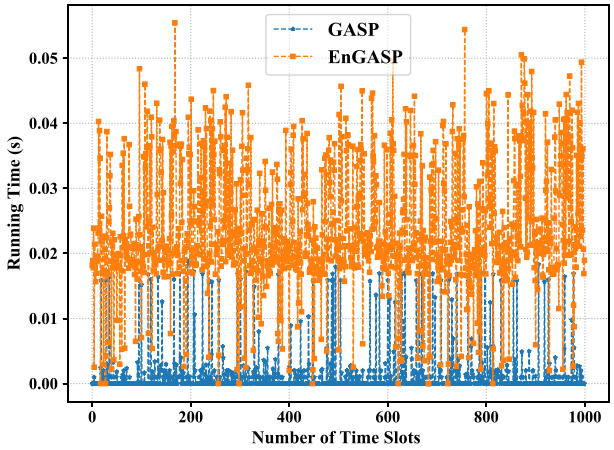


Fig. 4. Response time comparison with different time slots.

B. Simulation Results

First, we evaluate the performance of GASP and EnGASP with different time slots, respectively. The experimental result is shown in Fig. 3, where the x-coordinate denotes the time slots and the y-coordinate denotes the objective values corresponding to the problem $P2$. From this figure, several observations can be revealed as follows. First, as theoretically analyzed earlier, the performance of EnGASP is at least as good as GASP. The experimental result conforms to the theoretical analysis. Second, the performance of GASP at time slot 0 is the worst compared to other cases, which, however, is comprehensible and explicable as follows. According to (16), the calculation of $q(t+1)$ at time slot t depends on $q(t)$ and $E_t(\boldsymbol{\delta}^t)$. At the beginning of time slot (i.e., $t = 0$), we assume that all the vehicular applications are not cached at S. Thus, the objective value can be very large. Third, EnGASP helps improve the performance of GASP by two main stages (i.e., the filling step and the swapping step). Across different time slots, the two steps further optimize GASP to a certain extent.

Fig. 4 shows the change of the running time of GASP and EnGASP as the number of time slots increases. First, it is very clear that the running time of EnGASP is much larger than that of GASP. Most of the time, GASP can make a caching decision in real time, while EnGASP can make it within tens

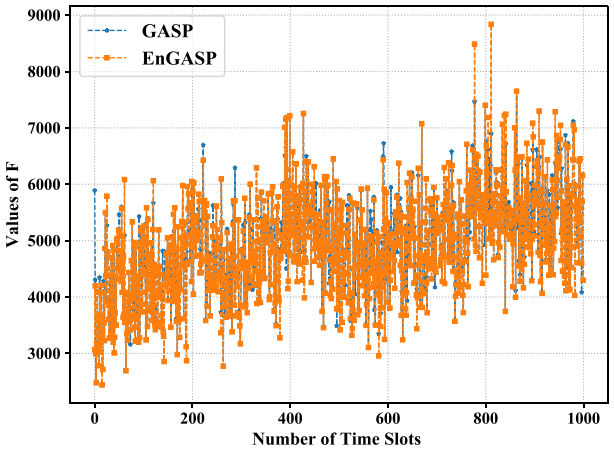


Fig. 5. $\mathcal{F}$ minimization with two heuristic rules.

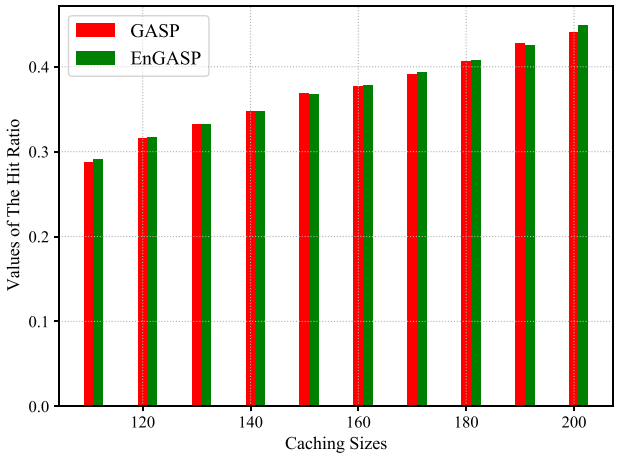


Fig. 6. Hit ratio comparison with different caching sizes.

of milliseconds. It is understandable since EnGASP needs the output of GASP as input parameters, i.e., EnGASP runs based on the caching decision obtained from GASP. As a result, the response time of EnGASP includes that of GASP, and thus, it is of polynomial time. Second, both GASP and EnGASP need complete accomplishment within each time slot. Furthermore, the tasks are generated randomly within each time slot, which means they are independent at different time slots. Therefore, the running time of both GASP and EnGASP do not increase as the number of time slots increases. Third, both GASP and EnGASP can make a caching decision in almost real time. It is of important significance to satisfy the strict time requirement of the vehicular applications.

Long-term energy constraint Q is converted into the per-slot energy constraint by the Lyapunov optimization technology, so we can pay attention to the per-slot problem optimization. GASP and EnGASP try to obtain the best caching decisions w.r.t. problem $P1$ by minimizing $\mathcal{F}$. The experiment has been conducted to evaluate the performance when minimizing $\mathcal{F}$, and the simulation result is shown in Fig. 5, where the x-coordinate represents the number of time slots and the y-coordinate represents the values of $\mathcal{F}$. It can be easily observed that EnGASP is better than GASP when the number of time slots is the same. Generally speaking, no matter how the number of time slots varies, EnGASP is better than GASP in most cases. On the other hand, the situation similar to the case shown in Fig. 3 happens when the time slot $t = 0$ and the reason is also the same as that in Fig. 3.

The metric hit ratio is an important performance indicator to evaluate the performance of the caching strategies in the information-centric networks. Furthermore, it is still applicable for evaluating the application-oriented caching strategies. In particular, Fig. 6 shows the variation of the hit ratio when GASP and EnGASP are applied under different caching sizes. In this experiment, the caching sizes range from 110 to 200 with a step of 10. It is obvious that the hit ratio increases for both GASP and EnGASP, as the caching sizes increase, this is because larger caching capabilities will enable more services to be cached. As expected, EnGASP is slightly better than GASP w.r.t. the hit ratio in most cases.

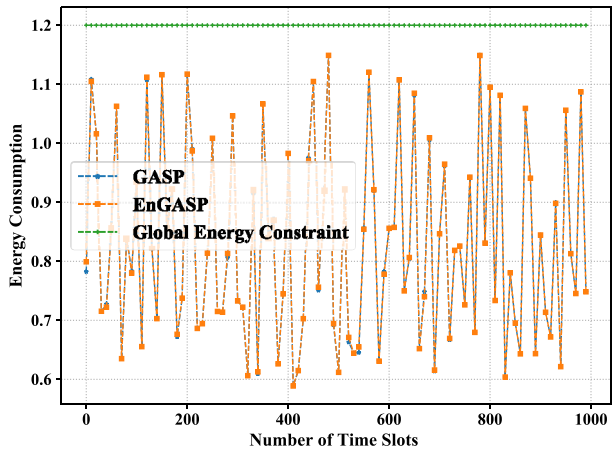


Fig. 7. Average energy consumption compared to the global energy constraint.

Another set of experiments is conducted to evaluate the average energy consumption at S across different time slots. More importantly, we need to check whether there are any constraint violations of energy consumption at different time slots. The simulation results are shown in Fig. 7. First, no matter how the number of time slots increases, the energy consumptions at S vary roughly from 0.5 to 1.2. The global energy constraint Q is 1.2, so there is no energy violation at all for the arbitrary number of time slots. Therefore, the experimental results accord with the theoretical analysis. Second, the energy consumption at S within each time slot depends upon the current caching decision profile. Generally, the more the number of applications, which are cached at S, the more the energy consumptions at S, for the reason that given a certain period of time, the running state of virtual environments should be maintained so as to avoid additional time overheads on the virtual initialization. The difference in the caching decision profile at different time slots leads to the different energy consumptions at different time slots.

As discussed earlier, some iteration-based evolutionary algorithms can also be adopted to obtain the optimal values of $\mathcal{F}$. In particular, we investigate the performance of these algorithms (e.g., GA and PSO) w.r.t. the capabilities to obtain

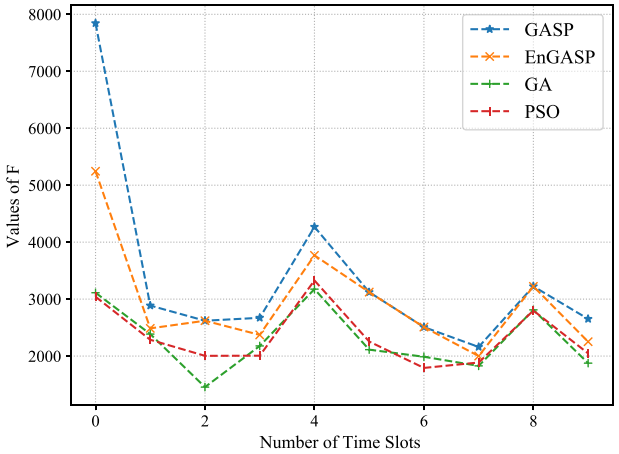


Fig. 8. Performance comparison with different number of time slots.

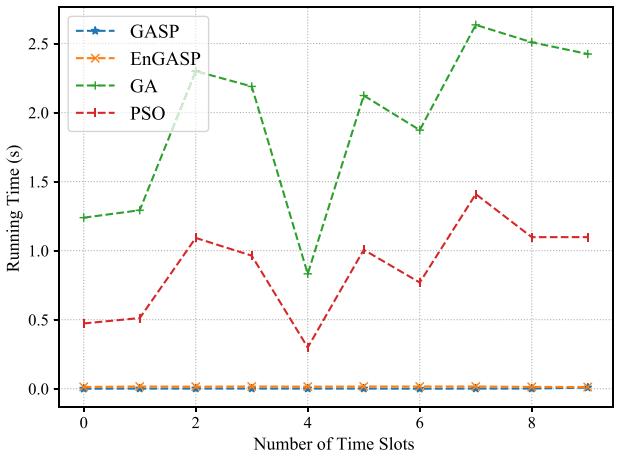


Fig. 9. Running time comparison with different number of time slots.

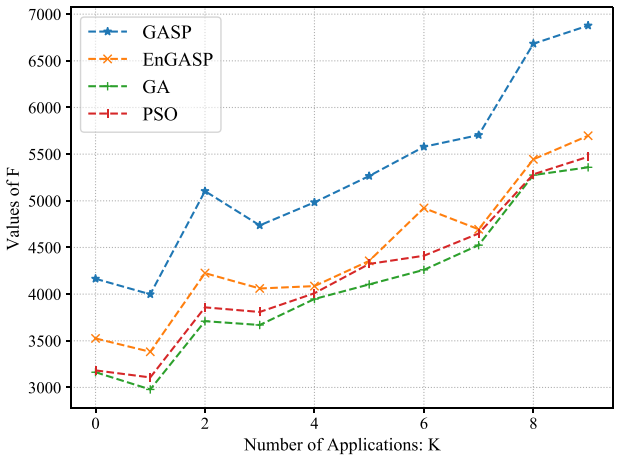


Fig. 10. Performance comparison with different number of applications.

the optimal values of $\mathcal{F}$ and the time complexity. The simulation results are shown in Figs. 8–11, respectively. Figs. 8 and 9 show the optimal values and the running time of the four approaches, respectively, when the number of time slots increases. Obviously, at least two conclusions can be drawn based on the observations. First, GA and PSO indeed have better capabilities to obtain the optimal values of $\mathcal{F}$ compared to our approaches GASP and EnGASP. Since the offloading

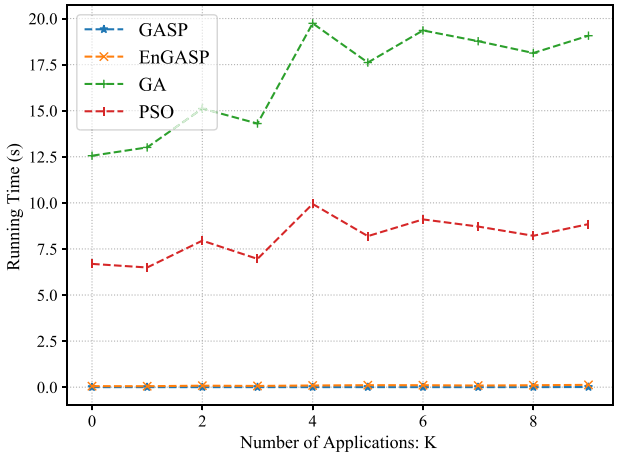


Fig. 11. Running time comparison with different number of applications.

requests are generated randomly in each time slot, and thus, they are independent in different time slots. The optimization gap between the evolutionary algorithms and our approaches is not becoming wider as the number of time slots increases. Second, the running times of GA and PSO are much longer than GASP and EnGASP. For example, it averagely takes about 2 s for GA to obtain the optimal value of $\mathcal{F}$, while it averagely tasks about 1.65 s for PSO to obtain the value of $\mathcal{F}$. GASP and EnGASP, on the other hand, can achieve the real-time response no matter how the number of time slots changes.

When we compare the four approaches under different numbers of applications, GASP and PSO are hardly acceptable despite the advantages over our approaches. As denoted in Fig. 11, both of GA and PSO take seconds to obtain the better values of $\mathcal{F}$. As a result, we can carefully draw a conclusion that the evolutionary algorithms are not as good as our approaches in the time-sensitive application scenario emphasized by this article. Our heuristic rules cater for the strict time requirements of vehicular applications at the expense of the precision of the solution, which, however, is acceptable to a great extent. In contrast, GA and PSO may display better performance with regards to precision, but they usually take subseconds or even seconds to achieve this goal. Such amount of time is unacceptable for the time-sensitive vehicular applications.

Next, we investigate the effect of V on the objective values obtained in LOCD. To be specific, GASP and EnGASP are adopted to solve problem $P2$ (line 5 in LOCD), respectively. As shown in Theorem 1, the caching decision obtained by solving $P2$ can be infinitely close to the solution to $P1$ by adjusting the variable V. The value of V varies from 800 to 1000, and the simulation results are shown in Fig. 12. From the figure, we can observe that LOCD with EnGASP has significant advantages over LOCD with GASP. On the other hand, owing to the random generation of tasks at different time slots, the best objective values are independent of each other across different time slots. Accordingly, it makes sense that the best objective values fluctuate a lot across different time slots.

The last set of experiments has been carried out to evaluate the performance of LOCD with the increasing number of

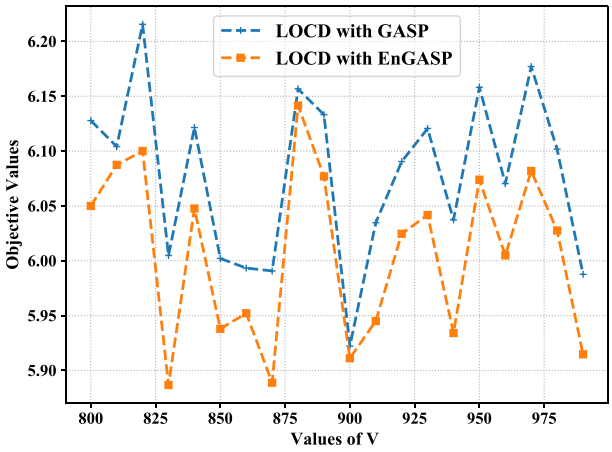


Fig. 12. Effect of V on the objective values.

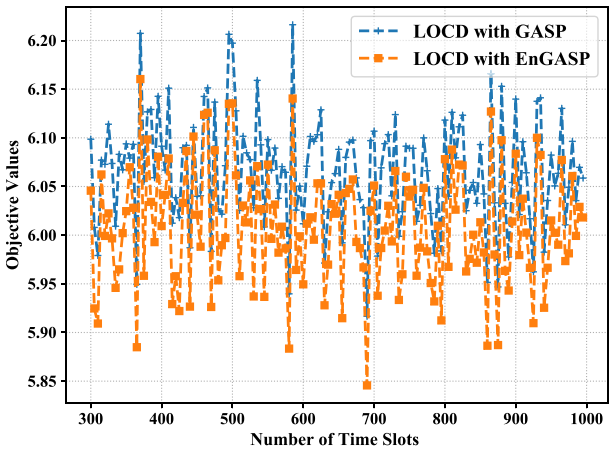


Fig. 13. Convergence investigation with different number of time slots.

time slots. As shown in Fig. 13, due to the global energy constraint Q, LOCD reveals its convergence either with GASP or EnGASP no matter how the number of time slots increases. For example, when the number of time slots is equal to 1000, the corresponding objective value is about 6.03. In spite of the fluctuation of these values across different time slots, this fluctuation has been confined to a limited interval (e.g., between 5.7 and 6.35). Furthermore, LOCD with EnGASP still outstands that with GASP all the time.

VI. Conclusion

VEC brings considerable benefits for vehicular applications in smart transportation. However, the performance of VEC is still challenged by high mobility of vehicles and limited communication resources, especially considering that the response time acts as the optimization objective during application outsourcing. In this article, we applied application caching to VEC to optimize the response time for the outsourced applications while satisfying the time slot spanned energy consumption. The Lyapunov optimization technology is adopted to convert the global energy constraint into per-slot energy constraints, so as to facilitate the response time optimization. Furthermore, two greedy heuristics are incorporated into the drift-plus-penalty-based algorithm for helping find the approximate optimal solution. We have evaluated the approach via a series of experiments. The simulation results reveal the advantages of our algorithms in terms of response time and energy consumption.

References

[1] J. Wang, C. Jiang, K. Zhang, T. Q. S. Quek, Y. Ren, and L. Hanzo, "Vehicular sensing networks in a smart city: Principles, technologies and applications," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 122–132, Feb. 2018.

[2] N. Zhang, S. Zhang, P. Yang, O. Alhussein, W. Zhuang, and X. S. Shen, "Software defined space-air-ground integrated vehicular networks: Challenges and solutions," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 101–109, Jul. 2017.

[3] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.

[4] Y. Yao, B. Xiao, W. Wang, G. Yang, X. Zhou, and Z. Peng, "Real-time cache-aided route planning based on mobile edge computing," *IEEE Wireless Commun.*, vol. 27, no. 5, pp. 155–161, Oct. 2020.

[5] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Comput. Netw.*, vol. 57, no. 16, pp. 3128–3141, 2013.

[6] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1473–1499, 3rd Quart., 2015.

[7] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 207–215.

[8] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoniem, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018.

[9] H. Xing, J. Cui, Y. Deng, and A. Nallanathan, "Energy-efficient proactive caching for fog computing with correlated task arrivals," in *Proc. IEEE 20th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, 2019, pp. 1–5.

[10] 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.

[11] G. Avino, M. Malinverno, F. Malandrino, C. Casetti, and C. F. Chiasserini, "Characterizing docker overhead in mobile edge computing scenarios," in *Proc. Workshop Hot Topics Container Netw. Syst.*, 2017, pp. 30–35.

[12] X. Hou *et al.*, "Reliable computation offloading for edge-computing-enabled software-defined IoV," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7097–7111, Aug. 2020.

[13] S. Li, S. Lin, L. Cai, W. Li, and G. Zhu, "Joint resource allocation and computation offloading with time-varying fading channel in vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3384–3398, Mar. 2020.

[14] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Intelligent resource allocation for utility optimization in RSU-empowered vehicular network," *IEEE Access*, vol. 8, pp. 94453–94462, 2020.

[15] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 570–584, Apr.–Jun. 2020.

[16] Z. Qin, S. Leng, J. Zhou, and S. Mao, "Collaborative edge computing and caching in vehicular networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2020, pp. 1–6.

[17] Y. Dai, D. Xu, K. Zhang, S. Maharjan, and Y. Zhang, "Deep reinforcement learning and permissioned blockchain for content caching in vehicular edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4312–4324, Apr. 2020.

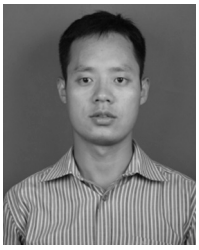
[18] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[19] M. Laroui, B. Nour, H. Moungla, H. Afifi, and M. A. Cherif, "Mobile vehicular edge computing architecture using rideshare taxis as a mobile edge server," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2020, pp. 1–2.

[20] C. Tang, X. Wei, C. Zhu, Y. Wang, and W. Jia, "Mobile vehicles as fog nodes for latency optimization in smart cities," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 9364–9375, Sep. 2020.

[21] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[22] Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, p. 60, 2019.
[23] D. Chen, Y.-C. Liu, B. Kim, J. Xie, C. S. Hong, and Z. Han, "Edge computing resources reservation in vehicular networks: A meta-learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5634–5646, May 2020.
[24] N. Cheng *et al.*, "Performance analysis of vehicular device-to-device underlay communication," *IEEE Trans. Veh. Technol.*, vol. 66, no. 6, pp. 5409–5421, Jun. 2017.
[25] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware IoT networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8262–8269, Oct. 2019.
[26] 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.
[27] X. Wei, C. Tang, J. Fan, and S. Subramaniam, "Joint optimization of energy consumption and delay in cloud-to-thing continuum," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2325–2337, Apr. 2019.
[28] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, no. 1, pp. 1–144, 2006.
[29] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, p. 211, 2010.



Chaogang Tang (Member, IEEE) received the B.S. degree from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2007, and the joint Ph.D. degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, Hong Kong, in 2012.

He is currently with the China University of Mining and Technology, Xuzhou, China. His research interests include mobile cloud computing, fog computing, Internet of Things, and big data.

Chunsheng Zhu (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of British Columbia, Vancouver, BC, Canada, in 2016.

He is an Associate Professor with the SUSTech Institute of Future Networks, Southern University of Science and Technology, Shenzhen, China. He is also an Associate Researcher with the PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen. He has authored more than 100 publications published by refereed international journals (e.g., IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, IEEE TRANSACTIONS ON CLOUD COMPUTING, *ACM Transactions on Embedded Computing Systems*, and *ACM Transactions on Cyber-Physical Systems*), magazines (e.g., *IEEE Communications Magazine*, *IEEE Wireless Communications Magazine*, and *IEEE Network Magazine*), and conferences (e.g., IEEE INFOCOM, IEEE IECON, IEEE SECON, IEEE DCOSS, IEEE ICC, and IEEE GLOBECOM). His research interests mainly include Internet of Things, wireless sensor networks, cloud computing, big data, social networks, and security.

Huaming Wu (Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2009 and 2011, respectively, and the Ph.D. degree (Highest Hons.) in computer science from Freie Universität Berlin, Berlin, Germany, in 2015.

He is currently an Associate Professor with the Center for Applied Mathematics, Tianjin University, Tianjin, China. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing, and deep learning.

Qing Li (Senior Member, IEEE) is currently a Chair Professor (Data Science) and the Head of the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. Formerly, he was the Founding Director of the Multimedia software Engineering Research Centre, and a Professor with the City University of Hong Kong, Hong Kong, where he worked with the Department of Computer Science from 1998 to 2018. Prior to these, he has also with the Hong Kong University of Science and Technology, Hong Kong, and the Australian National University, Canberra, ACT, Australia.

Prof. Li is the Chairperson of the Hong Kong Web Society and also served/is serving as an Executive Committee Member of IEEE-Hong Kong Computer Chapter and ACM Hong Kong Chapter. In addition, he serves as a Councilor of the Database Society of Chinese Computer Federation (CCF), a member of the Big Data Expert Committee of CCF, and is a Steering Committee Member of DASFAA, ER, ICWL, UMEDIA, and WISE Society. He is currently a Fellow of IET/IEE and a member of ACM-SIGMOD and IEEE Technical Committee on Data Engineering.

Joel J. P. C. Rodrigues (Fellow, IEEE) is a Professor with the Federal University of Piauí, Teresina—Pi, Brazil, and a Senior Researcher with the Instituto de Telecomunicações, Aveiro, Portugal. He has authored and coauthored over 780 publications in refereed international journals and conferences, three books, two patents, and one ITU-T Recommendation.

Prof. Rodrigues is the Leader of the Internet of Things Research Group (CNPq), Director for Conference Development—IEEE ComSoc Board of Governors, an IEEE Distinguished Lecturer, a Technical Activities Committee Chair of the IEEE ComSoc Latin America Region Board, and the Past Chair of the IEEE ComSoc eHealth and Communications Software TCs. He is the Editor-in-Chief of the *International Journal of E-Health and Medical Communications*.