Joint Optimization of Communication and Storage Latencies for Vehicular Edge Computing

Mostafa Kishani[1], Zdenek Becvar[1], Mohammadsaleh Nikooroo[1], Hossein Asadi[2]
[1]*Dpt. of Telecommunication Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague*
Prague, Czech Republic
[2]*Dpt. of Computer Engineering, Sharif University of Technology*, Tehran, Iran
{kishamos, zdenek.becvar, nikoomoh}@fel.cvut.cz, asadi@sharif.edu

***Abstract*—The latency associated with accessing data stored on edge computing servers for vehicles encompasses both the communication between a vehicle and a server as well as a latency of a data storage system. To enable low-latency vehicular services, an efficient resource management should consider the communication as well as the storage I/O cache resource allocation along with a data access pattern and a priority of individual vehicular services. Therefore, we focus on a joint optimization of communication and storage I/O cache resource allocation for access to data of vehicular services hosted by the edge computing servers. The proposed framework determines the data placement for the services and allocates communication and storage I/O cache resources to each service. The objective is to minimize the overall latency experienced by vehicular services for access to data. The edge computing platforms share storage and communication resources among various vehicular services, each having distinct priorities and data access rates or patterns. Hence, to reflect different priorities of services in resource allocation, our objective metric takes into account the service priority, data access frequency, and latency. We propose a feasible solution using dual relaxation considering both communication and storage latencies. The proposed solution reduces the average latency of vehicular services by up to 1.8x compared to the state-of-the-art resource allocation method for vehicular edge computing. Even more notable improvement is observed for high priority vehicular services, where the proposal leads to 2.5x lower latency compared to the state-of-the-art storage I/O cache architecture for virtualized cloud services.**

***Index Terms*—Vehicular Edge Computing, Latency, Communication, Storage, Mobile Edge Computing.**

I. Introduction

The autonomous vehicles impose transmission of a high volume of data among them-self and to/from an infrastructure of the communication network. To deliver delay sensitive vehicular services, low latency communication is required [1] together with processing of vehicular services in edge computing servers deployed close to the vehicles [2]. Apart from communication and computing, also data storage and access to data stored in edge computing servers should be optimized to guarantee a high quality of vehicular services supported by Vehicular Edge Computing (VEC) [3]–[5].

This project is supported by international mobility of research, technical and administrative staff of research organizations, with the project number *CZ*.02.2.69/0.0/0.0/18_053/0016980 and by the Ministry of Education, Youth and Sport of the Czech Republic under Grant LTT20004.

The performance of storage is recognized as a key bottleneck in VEC, since the current platforms fail to deliver the performance required for the desired autonomous vehicle's applications. A survey on the characteristics of edge computing applications [1] concludes that the autonomous vehicles generate over 100 GB data per vehicle per day and, thus, require a real-time connectivity with very low (sub-millisecond) latency. Supporting the features, such as real-time traffic monitoring, continuous sensing, or infotainment applications in the autonomous vehicles, motivates the use of vigorous data access/storage in VEC platforms [1], [6]–[8] while the existing platforms fail to address such performance requirements [1], [6]–[8]. Another key challenge is data movement, i.e., migrating a large amount of data from the storage of one edge computing server to another server for VEC [5], [9], [10]. Such data migration mandates an efficient management of both storage and memory resources. Previous investigations on constraints of the autonomous vehicles also conclude that both the computation and the storage are major bottlenecks in applications such as localization, object detection, and object tracking [5].

Another challenge in VEC is storage/communication resource allocation to the vehicular services with a wide range of priorities. The safety- and time-critical services, such as collision avoidance, have higher priority than the other services, such as software update [1]. Hence, the data access latency of such services should be taken into account and these applications should be prioritized. The VEC platforms share the storage subsystem among different vehicular applications despite the priority and storage demands. Hence, under an inefficient storage management, the data intensive applications, such as map/software update or sensing data upload, can also impact the storage performance of other applications, such as collision avoidance, resulting in a large and unacceptable data access latency for the safety-critical application.

The data access latency in VEC is a function of resources allocated for storage Input/Output (I/O) caching and communication. A joint allocation of the storage I/O caching and communication resources is, thus, crucial. However, due to very different nature of both types of resources, such optimization problem is Nondeterministic Polynomial-time (NP)-completed under realistic assumptions. Therefore, finding solution for such problem is complex if not impossible [11]. Note that the

storage I/O caching is different from the content caching [12], [13] and optimization of the storage I/O cache reduces the data access latency in a VEC regardless of the data content [4].

The main focus of works addressing the latency in VEC platforms is minimization of communication, computation, and network caching latencies. However, a considerable source of latency, the storage latency, is not addressed in these works. A group of previous works is concentrated on computation offloading [14]–[23], but only considering the computation processing using metrics such as Central Processing Unit (CPU) cycles, while storage as a significant source of latency is ignored.

Some other previous works concentrate on network caching and storage allocation in vehicular edge computing [12], [13], [24]. However, these works do not explore the storage access latency and do not investigate a potential reduction in the storage access latency via I/O caching optimization.

Data storage systems mainly rely on *Hard Disk Drives* (HDDs) and low-end *Solid-State Drives* (SSDs) as the main storage media to manage the storage services at a reasonable cost. However, these storage media have a relatively high latency. To provide the desired performance with a low latency, an efficient and widely used approach is the I/O caching. In the data storage system with I/O cache, the average storage latency of each application is directly affected by the size of I/O cache allocated to that application [25]–[29]. Hence, allocating the I/O cache resources is the subject of many studies, which target to reduce the storage latency in the edge and cloud computing systems [25]–[29]. In [25], the I/O cache for virtualized platforms is addressed, not considering the application priority. This architecture is proposed for a single storage system and is not suitable for multiple connected edge computing servers. A cache policy for virtualized platforms specialized for Virtual Desktop Infrastructures (VDI) and data warehousing is proposed in [26]. Another I/O cache architecture is proposed in [27] to address the demands of internet applications such as web or file services. S-CAVE solution developed in [28] is another I/O cache architecture that manages cache space among multiple virtual machines in commercial hypervisors. This method also relies on a single server and conclusions delivered in the paper are valid for applications such as web search or data warehousing. In [30], the authors propose an I/O cache allocation method for commercial virtualized platforms, relying on a single-server architecture. All these works are focused on general data center applications and are not designed for VEC platforms, hence, communication latency and resource allocation are neglected.

These works also do not consider the priority of services, which is important factor in VEC. We summarize the data storage challenges in VEC platforms not addressed in the previous works as follows: a) the latency of access to the storage subsystem is considerable, seriously contributing the overall latency of VEC services, while neglected in all previous works focused on VEC platforms, b) the storage architecture should be optimized for VEC applications while previous studies, which investigate storage architecture focus only on general data center applications, c) the previous studies neglect the priority of services, which is critical for VEC applications, d) both communication and storage latencies significantly impact the overall services latency, but these two aspects are not jointly addressed in the related works.

The storage I/O caching for the VEC platforms is investigated in our prior work [4]. Nevertheless, this prior work does not consider the communication latency and only targets the storage latency. The idea of a joint optimization of both storage and communication latencies is tackled in our another prior work [31], where we provide an initial investigation of both latencies and we demonstrate a significance of their joint optimization. To our best knowledge, exploration of data access in VEC platforms by jointly considering storage and communication latency is not addressed in any other work.

Hence, in this paper, we target the problem of joint optimization of the storage I/O cache and communication resource allocation for data access of vehicular services, such as collision avoidance, map/software download, or upload of sensed data, hosted by the edge computing platforms. The proposed algorithm allocates the unikernel to the edge computing server (base station) and determines the I/O cache size, bandwidth, and transmission power. To best of our knowledge, this is the first work on joint optimization of storage and communication latencies in edge computing platforms. The contributions of this work is summarized as follows:

- We propose novel algorithm minimizing the latency of access to data for vehicular services considering jointly communication as well as storage aspects. The algorithm places vehicular services to individual edge computing servers and allocates storage and communication resources for individual vehicular services. Moreover, service priority and access frequency are considered in allocation of storage and communication resources.
- Joint minimization of storage and communication latency is NP-Complete problem. Hence, we derive sub-optimal feasible solution using dual relaxation.
- Via simulations with realistic models, we demonstrate the proposed algorithm significantly reduces the overall latency of vehicular services stored in edge computing servers with respect to state-of-the-art-works.

The rest of paper is organized as follows. Section II presents the system model. In Section III, the problem addressed in this paper is formulated. Section IV elaborates the proposed solution. Then, in Section V, a performance evaluation setup is outlined and simulation results are discussed. Finally, Section VI concludes the paper.

II. System Model

In this section, we first present a general system overview. Afterwards, we elaborate models of the communication, storage, and overall latency in respective subsections.

A. General System Overview

We assume system consisting of V vehicles connected to K base stations. Each base station hosts one edge computing server serving the vehicle(s), as shown in Fig. 1. Therefore, there are K edge computing servers. Each vehicular service offloaded into the edge computing server appears in a form

of unikernels [32]. The unikernel is a lightweight, standalone machine image, used in the cloud computing environment due to its ability to improve performance, enhance security, simplify deployment, or large scalability. We consider a single unikernel for each service of the vehicle. For different services, such as for collision avoidance or map/software updates, each vehicle can be allocated with multiple unikernels, each handling a specific service. Considering multiple unikernels, each handling a single vehicle service has many benefits including higher security, more efficient resource allocation regarding the vehicle demands and priority, and easier mobility management. At the same time, each unikernel allocated for one vehicle is hosted in one edge computing server. To consider the unikernel to server allocation in the system model, we use the notation U_{v,k_i}, which represents the unikernel i from the edge computing server k allocated to the vehicle v. We also consider that each unikernel is served by only one edge computing server at a time. Hence, by considering N_k as the number of unikernels hosted by the edge computing server k, the total number of unikernels hosted by all edge computing servers is $U = \sum_{k=1}^{K} N_k$. Finally, we define the association matrix $A = \{a_{u,k}\}$, where $a_{u,k} \in \{0,1\}$ and $a_{u,k} = 1$ indicates the association of the u-th unikernel to the k-th edge computing server, i.e., placing the u-th unikernel into the k-th edge computing server. As each base station collocates one and only one edge computing server, matrix A determines the unikernel to base station placement.

In each base station and edge computing server, a hypervisor is responsible for monitoring of the services' behavior in a periodic manner, helping the base station and edge computing server in allocation of the communication resources and storage resources, respectively, to each unikernel. We define a monitoring period as the period for which the hypervisor monitors the services' behavior and the resources are allocated regarding the history of accesses in the previous monitoring period. The resource allocation is unchanged during the monitoring period and the allocation is revised at the end of each period. The length of the monitoring period Γ is a system model parameter. Γ is typically in order of few hundred milliseconds, while smaller Γ is required when we have fast changes in the workload behavior, i.e., workload characteristics including frequency of accesses, request size, access locality, and combination of read/write requests. The impact of monitoring period is evaluated in our simulations.

In the proposed VEC storage environment, each unikernel is characterized by three parameters:

- *Overall latency* of data request, D_{v,k_i}, is defined as the time between the data read/write request sent by the vehicle and the response from the edge computing server received by the vehicle. D_{v,k_i} includes communication as well as storage latencies.
- *Frequency* (number) of data requests, F_{v,k_i}, is the number of issued data storage accesses requests from the vehicle v to the unikernel i of the server k per unit of time. As the data accesses may be different in size, we consider a unit data access size of l_s bits and we convert the data access with an arbitrary size into multiple accesses with the size l_s.
- *Priority* $\rho_{v,k_i} \in \langle 0,1 \rangle$ is a static value indicating importance of the service (application) executed by the unikernel i, hosted by the server k, and allocated to the vehicle v. The priority is reflected in the optimization of both storage I/O cache and communication resources.

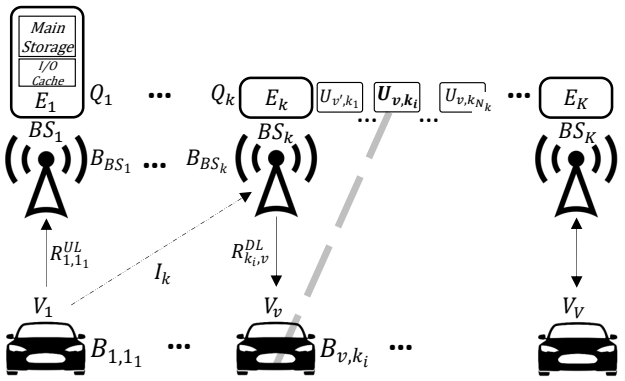

Fig. 1: Overview of the system model with unikernel i of edge computing server k, U_{v,k_i} associated to vehicle v (V_v).

Table I summarizes the variables and parameters defined in the systems model.

B. Communication

For the communication latency, downlink and uplink are taken into account.

1) Uplink communication latency: D^{UL}_{v,k_i} is defined as the latency between the vehicle v and the base station with the edge computing server k hosting the unikernel i of the vehicle v:

$$D^{UL}_{v,k_i} = \frac{l_{UL}}{R^{UL}_{v,k_i}} \tag{1}$$

where l_{UL} is the size of uplink data (in bits) represented by write data ($l_{UL} = l_s$) or read request ($l_{UL} = l_r$, where l_r is the

TABLE I: System Model Parameters

Parameter	Description
V	No. of vehicles
V_v	Vehicle v
K	No. of base stations, no. of edge computing servers
BS_k	Base station k
U	No. of unikernels
U_{v,k_i}	Unikernel i from edge computing server k, allocated to vehicle v
N_k	No. of unikernels hosted by the edge computing server k
N^V_v	No. of unikernels per vehicle
E_k	edge computing server k
B_k	Bandwidth of base station k
$B_{k_i,v}$	Bandwidth allocated to U_{v,k_i}
B^V_v	Bandwidth of vehicle V_v
P^T_k	Transmission power of base station k
$P^T_{k_i,v}$	Transmission power allocated to U_{v,k_i}
$P^T_{V_v}$	Transmission power of vehicle V_v
Q_k	I/O cache memory size of edge computing server k
$Q_{j,k}$	Size of I/O cache memory layer j of edge computing server k
$Q_{i,j,k}$	Size of I/O cache layer j allocated to U_{v,k_i}
D_{v,k_i}	Overall latency of unikernel U_{v,k_i}
F_{v,k_i}	Data access frequency of unikernel U_{v,k_i}
ρ_{v,k_i}	Priority of unikernel U_{v,k_i}
$\eta_{i,k}(s)$	I/O cache hit ratio of U_{v,k_i} for cache size s
$\Psi_{i,k}(s)$	Stack distance histogram of U_{v,k_i} at point s
$\Delta_{i,k}$	Maximum stack distance of U_{v,k_i}
Γ	Length of monitoring period

size of read request), and R^{UL}_{v,k_i} is the uplink data rate over the radio channel between the vehicle v and the base station k hosting the unikernel i. The data rate R^{UL}_{v,k_i} is defined as:

$$R^{UL}_{v,k_i} = B_{v,k_i} \times log_2(1 + \frac{P^R_{v,k_i}}{\sigma + I_k}) \quad (2)$$

where B_{v,k_i} is the communication bandwidth between the vehicle v and the base station k hosting the unikernel i associated with the vehicle v, P^R_{v,k_i} is the received signal power at the base station k hosting the unikernel i for the vehicle v, σ represents the thermal noise, and I_k is the interference to the base station k imposed by other concurrent communications.

2) Downlink communication latency: $D^{DL}_{k_i,v}$ is defined as the latency between the base station k collocated with the edge server with the unikernel i and the vehicle v served by the unikernel i:

$$D^{DL}_{k_i,v} = \frac{l_{DL}}{R^{DL}_{k_i,v}} \quad (3)$$

where l_{DL} is the size of downlink data (in bits) represented by downloading data ($l_{DL} = l_s$) or sending a write acknowledgment by the unikernel ($l_{DL} = l_a$, where l_a is the size of write acknowledgment). When the write request is issued from the vehicle to the base station, it is considered as "confirmed" by the vehicle upon receiving a write acknowledgment, indicating the data being received and stored on the VEC server. $R^{DL}_{k_i,v}$ is the downlink data rate between the base station k and the vehicle v served by the unikernel i. $R^{DL}_{k_i,v}$ is defined as:

$$R^{DL}_{k_i,v} = B_{k_i,v} \times log_2(1 + \frac{P^R_{V_{k_i,v}}}{\sigma + I_{k,v}}) \quad (4)$$

where $B_{k_i,v}$ is the communication bandwidth between the base station k and the vehicle v, $P^R_{V_{k_i,v}}$ is the power of the received signal at the vehicle v from the base station k hosting the unikernel i, and $I_{k,v}$ is the interference from all base stations (except the base station k) imposed to the vehicle v.

The communication power and bandwidth are allocated proportionally to the number of hosted unikernels, as also considered by previous works for the edge computing applications [33]–[35]. Hence, the bandwidth $B_{k_i,v}$ allocated for the communication between the base station k and the vehicle v to handle the unikernel i is $B_{k_i,v} = B_k/N_k$, where B_k is the bandwidth available at the base station k. In a similar way, the transmission power $P^T_{k_i,v}$ allocated for the communication from the base station k to the vehicle v for handling the unikernel i is calculated as $P^T_{k_i,v} = P^T_k/N_k$, where P^T_k is the total transmission power of the base station k. Similarly in each vehicle the bandwidth is $B_{v,k_i} = B^V_v/N^V_v$ and the transmission power is $P^T_{v,k_i} = P^T_{V_v}/N^V_v$, where N^V_v is the number of unikernels per vehicle, B^V_v is the bandwidth of vehicle V_v, and $P^T_{V_v}$ is the transmission power of vehicle V_v. Note that in practical applications, e.g., in mobile networks, the communication delay depends on modulation and coding scheme, which is technology dependent. Since we target a general and technology-independent solution, we stick to commonly used modeling of communication latency in line with related theoretical works [13], [16]–[18], [24], [36] and we do not go into details of modulation and network coding scheme.

C. Storage

Each edge computing server is equipped by a main storage and I/O cache resources, both managed by a hypervisor. Due to a relative affordability of the main storage compared to the other system components, the system designers usually tend to install a storage capacity considerably larger than what is defined for the system mission. Hence, we can assume that the size of the main storage resources demanded by the hosted unikernels is always smaller than the main storage capacity provisioned by the edge computing server. Therefore, this work focuses on an efficient communication and I/O cache resource management. Furthermore, we expect that storage I/O cache is allocated to the unikernels from local resources of the host server, like in [25], [30].

The main storage media is either mid-range SSD or HDD. The storage I/O cache is assumed to be composed of three layers of *Dynamic Random-Access Memory* (DRAM), *Non-Volatile Memory* (NVM), and high-end SSD. Due to the volatile nature of DRAM, the Read-Only policy [4] is conventionally considered for DRAM cache to prevent data loss in case of system failures. We consider the write-back policy [4], [37] for both NVM and SSD cache due to its performance superiority to the write-through policy. The rest of cache management policies are captured from the best-practices of the commercial I/O cache configurations and state of the art research projects, including *Least Recently Used* (LRU) cache replacement policy in all layers, exclusive management method in all cache layers, and a promotion policy that promotes all missed cache requests to DRAM cache [25], [30], [38]–[40]. We further assume that the main storage latency L_m, is higher than the cache memory latency L_c, as the media used in the cache memory (such as DRAM, NVM, and high-end SSD) performs better than HDD or mid-end SSD used in the main storage, in term of both latency and *Input/Output Per Second* (IOPS), by orders of magnitude [41], [42], hence:

$$L_c < L_m \quad (5)$$

The storage access is represented by either read or write requests. The write requests are written in the first cache layer, since these have typically similar latency not influenced by the I/O cache policy management [25], [30]. Hence, a constant storage write latency of $D^S_{W_{v,k_i}}$ is considered for all write requests in our model and we concentrate on optimizing the storage latency of read requests, $D^S_{R_{k_i,v}}$, using an efficient I/O cache resource allocation.

The average latency of storage read is a function of the cache hit ratio, the latency of cache memory, and the latency of main storage [4]. For the unikernel i serving the vehicle v and hosted by the edge server k considering a single-layer cache memory (extendable to multi-layer cache [4], [43]), the average latency of the storage read is:

$$D^S_{R_{k_i,v}} = \eta_{i,k}(Q_{i,j,k}) \times L_c + (1 - \eta_{i,k}(Q_{i,j,k})) \times L_m \quad (6)$$

where $\eta_{i,k}(Q_{i,j,k})$ is the hit ratio of cache memory with a size $Q_{i,j,k}$ for the unikernel i hosted by the server k.

We use *Hit Ratio Curves* (HRC) [44] to calculate the cache hit ratio η. Using stack distance analysis, HRC plots the hit ratio as a function of the cache size. A memory access to the address a is of a stack distance s if the number of s distinct addresses are accessed since the previous access to the address a. In the LRU replacement policy, the cache of size $s+1$ has a hit for all accesses with the stack distance s and lower. Let $\Psi_{i,k}$ be the stack distance histogram for individual memory addresses of U_{v,k_i}, then the Cumulative Distribution Function (CDF) of $\Psi_{i,k}$ at the point $Q_{i,j,k}-1$ gives $\eta_{i,k}(Q_{i,j,k})$ is:

$$\eta_{i,k}(Q_{i,j,k}) = \frac{\sum_{s=0}^{Q_{i,j,k}-1} \Psi_{i,k}(s)}{\Theta} \tag{7}$$

where Θ is the total number of storage accesses. If we define $\Delta_{i,k}$ as the maximum stack distance of the unikernel i at the edge computing server k, a cache size of $s=\Delta_{i,k}+1$ has the the hit ratio $\eta=1$ and is known as the ideal cache size. Note that the HRC is constructed only once in each monitoring period. Like in conveniently considered practical cases and in mainstream applications, we also assume that the hit ratio η is a concave function of the cache size (Q) if the optimum cache replacement policy, such as LRU, is employed [45]. Hence, the stack distance histogram Ψ is a monotonically non-increasing function of the cache size Q. The storage latency is not only a function of I/O cache size, but the waiting time also plays a role when a burst of storage requests is received. However, the I/O cache size is highly regarded as the most important factor affecting the average latency of the data storage systems [26], [28], [46], [47]. Moreover, the high-end SSDs/NVMs can handle multiple read/writes in parallel and provide almost constant latency when IOPS is under a specific threshold, allowing us to consider a constant latency for the I/O cache and concentrate on the I/O cache size optimization.

D. Overall Latency

The overall latency of the read request is defined as the time interval from sending the storage read request by the vehicle (e.g., downloading the map or road traffic status) to receiving data from the edge computing server by the vehicle. In the case of write requests (e.g., uploading the sensing data or reporting a collision incidence), the latency of storage request is the time it takes from sending the storage write data by the vehicle to receiving the write acknowledgment from the edge computing server at the vehicle.

As shown in Fig. 2, the overall latency $D_{R_{k_i,v}}$ of the storage read request by the vehicle v for the unikernel i at the edge computing server k is the sum of the uplink communication latency D^{UL}_{v,k_i} of the read request, the storage read latency $D^{S}_{R_{k_i,v}}$, and the downlink communication latency $D^{DL}_{k_i,v}$ for receiving the read data from the edge computing server by the vehicle . The latency of the storage write request $D_{W_{v,k_i}}$ from the vehicle v for the unikernel i hosted at the edge computing server k is defined as the sum of the uplink communication latency D^{UL}_{v,k_i} for sending the write data, the storage write latency $D^{S}_{W_{v,k_i}}$, and

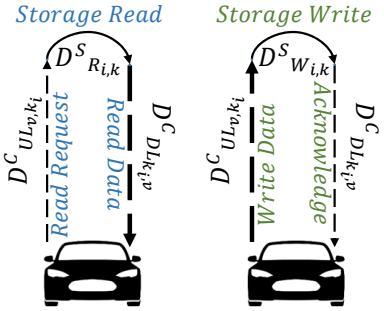

Fig. 2: Sequence of communication and storage latency upon read and write requests issued from the vehicle to the base station.

the downlink communication latency $D^{DL}_{k_i,v}$ for receiving the write acknowledgment by the vehicle.

The overall access latency D_{v,k_i} for the vehicle v served by the unikernel i hosted by the edge computing server k is:

$$D_{v,k_i} = \gamma_{v,k_i} \times D_{R_{k_i,v}} + (1-\gamma_{v,k_i}) \times D_{W_{v,k_i}} \tag{8}$$

where γ_{v,k_i} and $1-\gamma_{v,k_i}$ are the probabilities of read and write requests, respectively, generated by the vehicle v to the unikernel i hosted by the edge computing server k.

E. Handover Latency

When the vehicle moves to another base station, the allocated unikernel might be migrated to the new base station from the previous one depending on an applied mobility management algorithm. To reflect this aspect in our work, the migration penalty should be taken into account. To consider the effect of the migration process on the latency, we include the unikernel migration latency in the communication cost function. We assume optic fiber links among the base stations and the model of the migration latency is a function of the amount of migrated data, data transmission rate over the optic fiber link, and the distance between two base stations:

$$D^{M}_{k',k} = k_{disp} \times \frac{d'_{k',k}}{\Upsilon} + \frac{l_M}{R_f} \tag{9}$$

where l_M is the size of the unikernel which should be migrated (in bits), $d'_{k',k}$ is the distance between the source (k') and destination (k) base stations, R_f is the data transmission rate of the fiber-optic channel in bits per second, k_{disp} is the constant that accounts for the specific dispersion characteristics of the fiber, and Υ is the speed of light in the fiber (approximately 2/3 the speed of light in the vacuum [48]).

III. Problem Formulation

In this section, we present the targeted problem of the minimization of the overall latency of services provided to vehicles considering both communication and storage aspects. The objective metric should take all three unikernel characteristics we concern about, i.e., priority of the application ρ, frequency of the application data requests F, and latency D, into account (see Section II-A). The objective metric is defined as the multiplication of these three characteristic, $\rho \times F \times D$.

We consider the product of these three characteristics in our objective metric, so that all parameters directly impact the objective metric. Out of these three parameters, the unikernel priority and the unikernel data access rate are determined and are not affected by our design criteria. However, the communication and storage latencies are impacted by our design criteria and directly impact the objective metric. Hence, we discuss how each parameter affects our objective. We also elaborate the impact of the design criteria on the storage and communication latencies, and consequently, the objective metric.

Unikernel priority, ρ: The unikernel with a higher priority has a higher impact on the objective metric. Hence, if two unikernels have the same data access rate but different priorities, improving the latency of the higher priority unikernel results in a more notable improvement in the objective metric. This way, the proposed solution provides a lower latency to high-priority services.

Unikernel data access rate, F: The unikernel with a higher data access rate has a higher impact on the objective metric. Hence, if two unikernels have the same priority but different data access rates, improving the latency of the unikernel with higher access rate results in a more notable improvement in the objective metric. This way, the proposed solution provides a lower latency to the high-access rate services. The motivation for an inclusion of F in our objective metric is that by omitting F, we may allocate a relatively large amount of resources to the unikernel that has a low number of data accesses. This unfair resource allocation would result in a performance imbalance between the two unikernels with the same priority. For example, consider the unikernels A and B have the same priority. If the frequency of the unikernel B is 1000 times higher than the frequency of A (while the rest of parameters are the same), in the case of omitting F, we would allocate the same amount of resources to both A and B. Then, the unikernel A would significantly outperform B in terms of the overall latency of accessing data, as well as the latency of accessing a single request, since the unikernel A would be granted with the same amount of resources as B to respond only 1 request instead of 1000 requests responded by the unikernel B.

Unikernel overall data access latency, D: Defined as the time between the data read/write request sent by the vehicle and the response from the serving unikernel received by the vehicle, the overall data access latency is modeled as the aggregation of communication and storage latency. Hence, by reducing the latency, we directly reduce the objective metric. This way, the resource allocation resulting in a lower latency is encouraged.

In the following, we describe how each design criteria impacts the communication and storage latencies:

- **Storage Latency:** The storage latency is impacted by the cache hit ratio and the read/write ratio of the unikernel workload. The read/write ratio is determined and is not impacted by our design criteria. However, the cache hit ratio is impacted by the allocated cache size, which is the design criteria in our proposal, as indicated in the system model.
- **Communication Latency:** Communication latency is impacted by the association of the unikernels (vehicles) to the base stations (edge computing servers). This association impacts the communication latency in two ways: 1) Allocating the unikernel to a far base station results in a lower data transmission rate under the same bandwidth, transmission power, and noise conditions. 2) Sharing the communication resources (bandwidth and transmission power) of the base station between a larger number of unikernels results in a lower share of bandwidth and transmission power allocated to each unikernel, resulting in a lower data transmission rate under the same distance and noise conditions.

To reflect all these aspects, the problem is formulated as:

$$\min \sum_{k=1}^{K} \sum_{i=1}^{N_k} \rho_{v,k_i} \times F_{v,k_i} \times D_{v,k_i} \tag{10}$$

$$s.t. \quad \sum_{i=1}^{N_k} Q_{i,j,k} \leq Q_{j,k}, \quad \forall k \in \langle 1,K \rangle, \forall j \in \langle 1,\beta \rangle \quad (a)$$

$$\sum_{j=1}^{\beta} Q_{i,j,k} \leq \Delta_{i,k} + 1, \quad \forall k \in \langle 1,K \rangle, \forall i \in \langle 1,N_k \rangle \quad (b)$$

where N_k is the number of unikernels in the edge computing server k, β is the number of cache layers, and $Q_{j,k}$ is the size of the cache memory layer j in the edge computing server k.

The constraint *(a)* in (10) limits the the allocated cache memory from the layer j of the edge computing server k to the size of the cache memory installed in the edge computing server k. The constraint *(b)* guarantees the aggregation of cache space allocated to the unikernel i from different cache layers is within the limit implied by the ideal cache space determined by the maximum stack distance, $\Delta_{i,k}$. Potential constraints related to the communication bandwidth and power are guaranteed to be satisfied regarding the system model.

IV. Proposed Solution

In this section, we propose a solution addressing the association of unikernels to the edge computing servers, as well as I/O cache size allocation.

A. General Principles of the Solution

A joint optimization of cache size and association of unikernels to the edge computing servers, as defined by matrix A (Section II-A), is challenging mainly due to the following facts: *i)* the objective is non-convex with respect to cache size, according to (7), and *ii)* the discrete nature of the association A (unikernel to edge computing server) makes the solution non-tractable. To tackle this, we propose a sub-optimal solution using dual relaxation. In particular, we first determine placement of the unikernels to edge computing servers A, resulting to minimum *communication objective metric*, i.e., the minimum objective metric while ignoring the storage latency and only considering the communication latency, as presented in Section IV-B.

After determining the placement of the unikernels to edge computing servers A, we find the efficient I/O cache size allocated to each unikernel in Section IV-C. To this end, we first relax the cache size constraint on each server (constraint (a) in (10)) and replace it with a constraint on the overall cache size allocated from all edge computing servers. In the

relaxed problem, the cache allocation is always possible from the edge computing server assigned by matrix A, resulting to the minimum communication objective metric. For this relaxed problem, we propose the solution determining the optimal I/O cache size allocated to each unikernel, minimizing both communication and storage objective metrics.

The optimum cache size allocation in the relaxed problem can, however, result to a possibly non-feasible solution to the main problem (10), as the cache size constraint (a) in (10) may be violated in some edge computing servers, as we discuss in Section IV-D. We show the aggregation of over-allocated cache in violated servers is equal to the aggregation of free cache space in the rest of servers. Finally, we propose a direct heuristic algorithm to achieve a feasible, but sub-optimal allocation. This algorithm replaces some services from the violated edge computing servers into the edge computing servers with free I/O cache space, if this replacement improves the objective metric.

B. Unikernel Placement Problem

In this section, we find a placement of unikernels to edge computing server (collocated with base stations) resulting to the minimum communication objective metric. As the communication resources are managed by the base station collocating the edge computing server and regarding the system model, each base station collocates one and only one server, the unikernel to edge computing server association is determined by the unikernel to base station association. Regarding the system model, the storage latency D^S_{v,k_i} is only a function of cache size. Hence, we remove the storage latency from the objective metric in (10) and only keep the communication latency. Hence, the goal is to find a solution to the following problem:

$$\min \sum_{k=1}^{K}\sum_{i=1}^{N_k}\rho_{v,k_i}\times$$

$$(F_{v,k_i}\times\frac{l}{\frac{B_k}{N_k}\times log_2(1+\frac{\frac{P_k^T}{N_k}(\frac{\lambda}{4\pi d_{i,k}})^2}{N+I})}+k_{disp}\times\frac{d'_{k',k}}{\Upsilon}+\frac{l_M}{R_f}) \quad (11)$$

The primary placement problem is a weighted bipartite graph matching in the unikernels part, as shown in Fig. 3. To solve such problem, we propose the solution shown in Algorithm 1.

Algorithm 1 Unikernel placement

1: **for** $i \leftarrow 1$ to U **do**
2: **for** $j \leftarrow 1$ to $\frac{U}{K}$ **do**
3: **for** $k \leftarrow 1$ to K **do**
4: $temp[i][(j-1)\times K+k] =$
5: $\rho_i \times F_i \times \frac{l}{\frac{\frac{P_U^T}{K}(\frac{\lambda}{4\pi d_{i,k}})^2}{\frac{B}{K}\times log_2(1+\frac{N+I}{N+I})}} + k_{disp}\times\frac{d'_{k',k}}{\Upsilon}+\frac{l_M}{R_f})$
6: **end for**
7: **end for**
8: **end for**
9: Hungarian (temp)

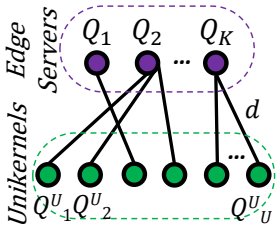

Fig. 3: Cache placement problem as a weighted bipartite graph, matching in the unikernels part.

In the proposed algorithm, we turn the graph shown in Fig. 3 into a bipartite graph with the number of U vertices in each part, recognized as a Balanced Assignment problem. To this end, in the bipartite graph of Fig. 3, we repeat each base station $\frac{U}{K}$ times (line 2), and construct the weight matrix $temp$ for the new bipartite graph with the number of U vertices in each part. The weight of each edge of the graph is calculated considering the communication objective metric (line 4). Finally, using Hungarian algorithm, we find the perfect matching of unikernels U_{temp} to the base stations K, resulting in the minimum weight. This algorithm allocates the same number of unikernels ($\frac{U}{K}$) to all base stations.

C. Constraints Relaxation and optimum cache allocation

In the previous section, the solution for the unikernel to base station placement, represented by matrix A, is proposed. In this section, we propose an algorithm for cache size allocation to each unikernel. As the unikernel to base station placement is already done by considering the communication objective metric, we remove the communication latency from the objective metric (10). To solve the problem, we first relax the cache size constraint on each edge computing server, assuming the efficient cache size can be allocated from any edge computing server, as far as the overall allocated cache does not exceed the aggregated cache sizes available on the edge computing servers. This relaxation helps us to find the optimal I/O cache size for the optimal unikernel to base station placement, resulting a lower bound for the main problem (10). In the relaxed problem, we find the optimal cache size considering the constraint limiting the total cache size allocated to the unikernels to the aggregation of the cache space installed throughout the edge computing servers, i.e., assuming:

$$\sum_{i=1}^{U} Q_i^U = \sum_{k=1}^{K} Q_k \quad (12)$$

where Q_i^U is the cache size allocated to the unikernel i. This relaxation allows us to replace the constraint (a) in (10) with $\sum_{k=1}^{K}\sum_{i=1}^{N_k} Q_{i,j,k} \leq \sum_{k=1}^{K} Q_{j,k}$. As such, the cache allocation is always possible from the edge computing server whose base station is determined by matrix A, resulting to the minimum communication objective metric. The relaxed problem is then

formulated as:

$$\min \sum_{k=1}^{K}\sum_{i=1}^{N_k} \rho_{v,k_i} \times F_{v,k_i} \times D_{v,k_i} \tag{13}$$

$$s.t. \quad \sum_{k=1}^{K}\sum_{i=1}^{N_k} Q_{i,j,k} \leq \sum_{k=1}^{K} Q_{j,k}, \quad \forall j \in \langle 1,\beta \rangle \quad (a)$$

$$\sum_{j=1}^{\beta} Q_{i,j,k} \leq \Delta_{i,k}+1, \quad \forall k \in \langle 1,K \rangle, \forall i \in \langle 1,N_k \rangle \quad (b)$$

Proposition 1. *The optimal solution of (13) is a lower bound to the optimal solution to (10).*

Proof. Suppose m is the optimal solution to the main problem (10). Hence, m satisfies the constraint (a) in (10). By aggregating the constraint (a) in (10) for different edge computing servers (different k values), we obtain the constraint (a) in (13). Hence, m is the feasible solution to (13). Consider m' is the optimal solution to (13). As m' is the lower bound to all feasible solutions to the problem (13), m' is also the lower bound to m, i.e., to the optimal solution to (10). □

In Algorithm 2, the cache allocation is optimized for the fixed unikernel to base station placement, already determined by Algorithm 1. We consider zero cache allocation for the initial state (line 1 and line 2) and we allocate the cache chunks one-by-one until all chunks are allocated (line 4 to line 10). Per iteration of the while loop (line 4), we allocate a single cache chunk. To this end, we find the unikernel i with the highest *objective achievement*, $\alpha_i = \rho_i \times F_i \times \Psi_i(Q_i)$, defined as an increase in the objective metric when allocating one extra cache chunk to the unikernel (line 5). Let Q_i be the current number of cache chunks allocated to the unikernel i and $\Psi_i(Q_i)$ be the stack distance histogram of i at the point Q_i. Regarding the definition of the stack distance, $\Psi_i(Q_i)$ is an increase in the hit ratio caused by the cache size increase from Q_i to Q_i+1. For example, when the cache size increases from 0 to 1, the hit ratio increases by $\Psi_i(0)$. Let $\eta_i(Q_i)$ be the current hit ratio of the unikernel i and $\eta_i(Q_i+1)$ be the hit ratio of i after allocating one extra cache chunk to it; then, we have:

$$\eta_i(Q_i+1) = \eta_i(Q_i) + \Psi_i(Q_i) \tag{14}$$

We allocate one cache chunk from the edge computing server e determined by matrix A to the unikernel i with the maximum objective achievement (line 5). Afterwards, we increase $Q_{i,e}$ (representing the number of cache chunks allocated from the edge computing server e to the unikernel i) and Q_i^U (representing the number of chunks allocated to the unikernel i) to track the number of cache chunks allocated to each unikernel (line 8 and line 9), As we assume allocating cache to the unikernel only from one edge computing server, $Q_i^U = Q_{i,e}$.

Algorithm 2 Optimum solution to the relaxed problem (13)

1: **for** $i \leftarrow 1$ to U **do**
2: $\quad Q_i^U = 0$
3: **end for**
4: **while** $\sum_{j=1}^{U} Q_j^U \leq \sum_{k=1}^{K} Q_k^E$ **do**
5: $\quad i = max(\rho_r \times F_r \times \Psi_r(Q_r^U)), r \in \langle 1,U \rangle$
6: $\quad$ **if** $Q_i^U < \Delta_i + 1$ **then**
7: $\quad\quad e = min(A(r)), k \in \langle 1,K \rangle$
8: $\quad\quad Q_{r,e} = Q_{r,e} + 1$
9: $\quad\quad Q_r^U = Q_r^U + 1$
10: **end while**

Proposition 2. *The cache chunk assignment following the proposed algorithm is optimal.*

Proof. Please see Appendix. □

D. Derive a feasible solution

Due to the previous relaxation, the cache size constraint can be violated in some edge computing servers (due to cache space over-allocation), while some edge computing servers have free unallocated cache space. As the previous cache allocation problem assumes the overall cache allocated to the unikernels is equal to the overall cache installed on the edge computing servers, asserted in (12), the aggregation of over-allocated cache space (the cache space allocated in an edge computing server over its installed cache size) is equal to the aggregation of unallocated cache space in the rest of edge computing servers.

To derive a feasible cache allocation with the constraint (a) in (10), we define the Replacement Cost Coefficient of the unikernel i, $\theta_i = F_i \times P_i$, as the product of the access frequency and priority of the unikernel i and we start replacing the unikernels with smallest θ to the edge computing server with free cache space, until the cash size constraint violation is removed. To this end, using exhaustive search, we find the edge computing server with free cache space resulting to the maximum improvement in the objective metric. With the unikernel replacement, the number of unikernels in the source base station e_1 and destination base stations e_2 changes. Regarding the equal splitting of communication bandwidth and power in the system model, an update in the number of unikernels in each base station changes the bandwidth and power allocation, considered in our analysis. Replacing unikernel u_1 with the smallest θ from server e_1 to server e_2 is taken only if it improves the objective metric. In case no server e found that replacement of unikernel u_1 from e_1 to e improves the objective metric, we fix the current unikernel placement in server e_1 and stop replacement. However, in this case the cache size constraint is still violated in server e_1. To remove this violation, we reallocate the I/O cache resources of server e_1 to the unikernels placed in server e_1, using Algorithm 2.

In practice, there is a possibility of *internal fragmentation* after running the re-placement algorithm. The internal fragmentation is understood as the free cache space in the edge computing server that is too small be allocated to any unikernel in the edge computing servers with an over-allocated cache space. As an example shown in Figure 4, suppose we have the edge computing server e_1 with 10 over-allocated cache chunks, while the edge computing servers e_2 and e_3 have 8 and 2 free cache chunks, respectively. Suppose that the unikernel u_1 of e_1 with the optimum cache size of 9, has the smallest θ. The algorithm starts re-placing the unikernels from the one with the smallest θ, i.e., u_1 in this example. However, the algorithm fails to finish the replacement of remaining unikernels in this

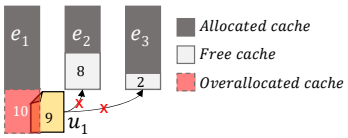

Fig. 4: Internal fragmentation problem in cache placement

example, since u_1 cannot be allocated to either of e_2 and e_3. In this case, we take one of two following decisions that results in a smaller objective metric value: a) re-place u_1 to the edge computing server with the largest free cache space, i.e., to e_2 in our example, and reallocate the I/O cache resources of server e_2 to the unikernels placed in server e_2, using Algorithm 2, or b) keep u_1 in the edge computing server where it is already placed, i.e., e_1 in the example, and reallocate the I/O cache resources of server e_1 to the unikernels placed in server e_1, using Algorithm 2.

V. Performance Analysis

In this section, we present simulation setup and the performance of the proposed framework in terms of average latency.

A. Simulation Setup

Fig. 5 shows the overview of our experiment flow. To evaluate the proposed framework, we post-processes the real block-layer traces of the edge computing server serving the autonomous vehicles. To model vehicle's movement, we capture vehicle trajectory from highD dataset [49] and we use the whole traffic data of highD dataset (16.5 hours) with 1/25 second timescale.

We define four vehicular services including collision avoidance, sensing data upload, map/update download, and other general type of services described as follows (and also detailed in Table II):

- **Collision avoidance** workload is considered as a high-priority and real-time service. This workload has combination of small (32 kb) read/writes (70%/30%) with relatively high temporal locality (achieved by Zipf 1.2 distribution).
- **Sensing data upload** from the autonomous vehicle to the edge computing server. This service is of a normal (medium) priority and generates sequential big (32768 kb) writes in the edge computing server.
- **Map and software update** sent from the edge computing server to the autonomous vehicles. Also, this service is of a normal priority, but generates sequential big (32768 kb) read requests in the edge computing server.
- **Other services for autonomous vehicles** with normal priority and with random distribution of accesses with high locality (achieved by Zipf 1.2 distribution).

We conduct the simulations with five representative workloads of collision avoidance (referred as *Collision* in the results), sensing data upload (*Sensing*), map and software update (*Update*), other services for autonomous vehicles (*Other*), and a mix of all services with equal probability of each out of four basic service types (*Mix*). The basic services are generated using Flexible I/O (FIO) benchmarking tool [50]. The storage block

TABLE II: Characteristics of autonomous vehicle service types

	Collision Avoidance	Sensing Data Upload	Map/Software Update	Other
Read/Write %	70/30	0/100	100/0	70/30
Request Size	32 kb	32768 kb	32768 kb	32 kb
Access Pattern	Zipf 1.2	Sequential	Sequential	Zipf 1.2
Priority	Real-time	Normal	Normal	Normal

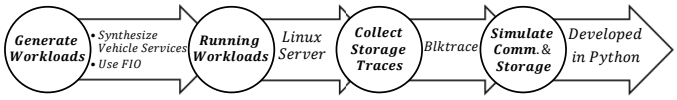

Fig. 5: Experiment flow

accesses for each service are collected using Linux *blktrace* tool [51].

We simulate the vehicular edge computing with the system model parameters summarized in Table III. We consider each edge computing server is collocated with one base station deployed along the road with equal inter-site distance of 1000 m To model vehicle's movement, we capture highway tracks data from highD dataset [49]. We also consider the communication bandwidth and power are allocated by the transmitting device proportionally to the number of unikernels hosted by the edge computing server of the base station serving the vehicle, as detailed in the system model, considering the carried frequency of 2.6GHz. We simulate the storage subsystem considering a 3-level cache which consists of DRAM, NVM, and SSD. The latency of each cache layer, as well as main storage (HDD) is summarized in Table III. We consider a high priority for collision avoidance services by setting $\rho = 1$ and normal priority for the rest of vehicular services with $\rho = 0.5$. We report the average latency, defined as the aggregation of communication and storage latency (8).

As no previous work considers both communication and storage latency together, we compare the proposed framework with three state-of-the-art platforms, including two I/O cache architectures for edge and virtualized platforms, *Priority-Aware Block Data Storage Architecture for Edge Cloud Serving Autonomous Vehicles* (PADSA) [4] and *Efficient Two-level I/O Caching Architecture for Virtualized Platforms* (ETICA) [25], and a workload scheduling platform for VEC which considers the communication and computation latencies, *Workload Scheduling in Vehicular Networks With Edge Cloud Capabilities* (WSVN) [36]. All three previous works consider a single edge computing server, while PADSA and ETICA do not take the communication latency into account. To be able

TABLE III: Simulation Parameter Values

Notation	Parameter	Value
K	Number of edge computing servers	10
U	Number of unikernels (4 unikernels per vehicle)	1000
L_c (DRAM)	I/O cache memory latency	7.9 ns
L_c (NVM)	I/O cache memory latency	70 ns
L_c (SSD)	I/O cache memory latency	25 us
L_m	Main storage latency	10 ms
P^T	Transmit power of edge computing server base station	46 dBm
P_i^T	Transmit power of vehicle	20 dBm
l	Size of unit storage access	32 kb
σ	Noise power	-90 dBm
B	Bandwidth of edge computing server base station	100 MHz

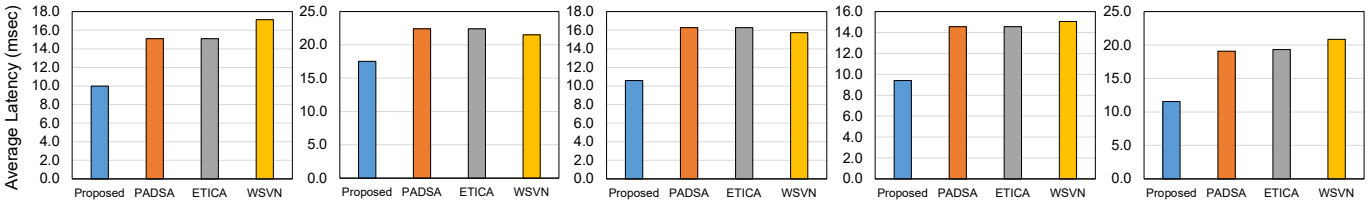

Fig. 6: Aggregation of communication and storage latency for: a) Collision Avoidance, b) Sensing Data Upload, c) Map/Software Update, d) Other, and e) Mix workloads.

to compare performance, we consider PADSA and ETICA allocate unikernels to the edge computing server with free cache space whose base station has the lowest distance from the vehicle, assuming that the allocation is performed by order, started from the unikernel 1 and finished with the unikernel U. WSVN considers communication latency of request upload and response download, as well as computation latency, but no storage latency. To be able to compare performance, we consider high-priority vehicle services (e.g., collision avoidance) as what considered as *Delay-Intolerant Task* in WSVN. We also consider each storage request in our system model as a computation request in WSVN, by considering that both data size (in bits) and computation requirement (CPU cycles) in WSVN are proportional to the storage access size in our system model.

Conducting Algorithm 1 is of time complexity $O(U \times K^3)$ (U for the main loop and K^3 for the Hungarian algorithm), where U is the number of unikernels and K is the number of base stations. Constructing Algorithm 2 is of time complexity $O(Q + U \times (R \log Z))$ where $Q = \sum_{k=1}^{K} Q_k$ is the total number of cache chunks in all edge computing servers, R is the total number of storage requests (length of workload) in the previous monitoring window, Z is the number of unique references (addresses) [44] in the previous monitoring window, and $R \log Z$ is the time complexity of constructing the stack distance histogram Ψ and hit ratio curve η for each unikernel. Finally, deriving a feasible solution after running Algorithm 1 and Algorithm 2 is of time complexity $O(U \times K)$. Hence, the whole solution is of time complexity $O(U \times (K^3 + R \log Z) + Q)$

B. Results

Fig. 6 compares the average latency of the proposed framework with the state of the art works. We show the average overall latency of all services, including both high-priority and normal services. The proposal improves the average latency compared to the state of the art works in all examined workloads in order of 23%-80%, thanks to an efficient service placement and considering the priority of services, reducing both communication and storage latency. Note that PADSA and ETICA perform the same in terms of communication latency and their performance difference is due to the storage latency. We observe the most notable latency improvement in the Mix workload. This observation is justified by the fact that the Mix workload has the combination of high-priority and normal-priority services and benefits more from the proposed framework, which considers the priority of services in the objective metric.

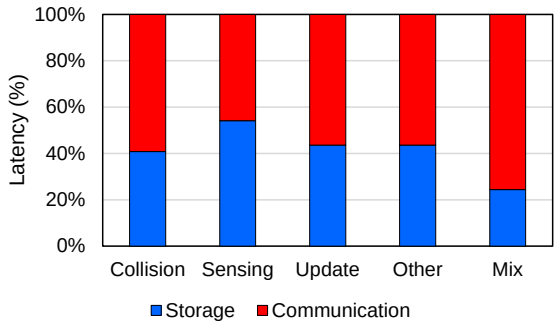

Fig. 7: Share of communication and storage in overall latency.

In Collision, Sensing, Update, and Other workloads, we only have one type of vehicular service. Hence, both PADSA and ETICA perform similar in terms of storage latency, as the major benefit of PADSA over ETICA, which is considering the service priority, does not show off when we have only one service type. The performance superiority of the proposed method in Collision, Sensing, Update, and Other workloads is also described by the more efficient storage and communication resource management, regardless of the priority of the services. In Sensing and Update workloads, WSVN performs slightly better than PADSA and ETICA. This observation is described by the fact that Sensing and Update workloads have sequential access pattern. Sequential workloads do not benefit I/O cache. Hence, all requests of Sensing and Update workloads are handled by accessing the main storage media, resulting the similar storage performance for all examined methods. The lower latency of the proposed method and WSVN compared to PADSA and ETICA is due to the more efficient allocation of communication resources.

Workload Collision and workload Other both have random read/write access pattern. Hence, both benefit an efficient I/O cache resource allocation which reduces the storage latency. This fact describes the relatively high latency of WSVN, as it has a high storage latency. The proposed method performs similar to PADSA and ETICA in terms of storage latency when we do not have services with different priority, while the better latency of the proposed method in workload Collision and Other is described by the more efficient communication resource allocation, resulting to lower communication latency.

Fig. 7 shows the a share of the communication and storage

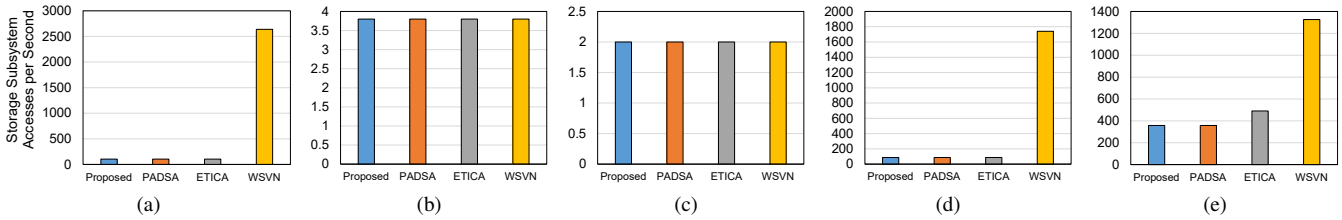

Fig. 8: Number of accesses to the storage subsystem per second (the average of whole runtime) for: a) Collision Avoidance, b) Sensing Data Upload, c) Map/Software Update, d) Other, and e) Mix workloads. The lower number shows better cache performance.

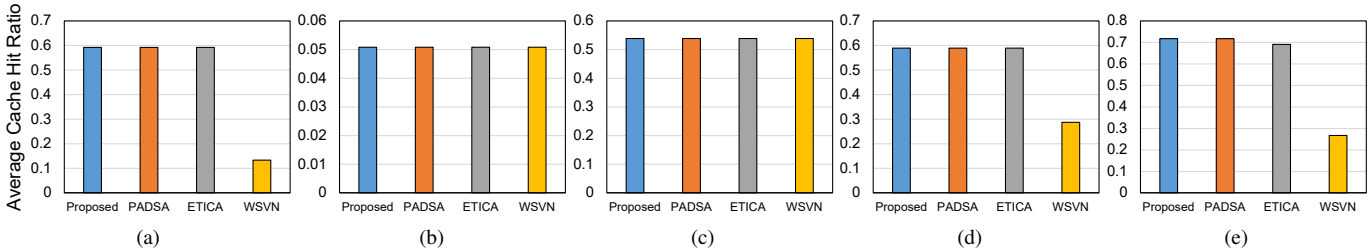

Fig. 9: Average cache hit ratio for: a) Collision Avoidance, b) Sensing Data Upload, c) Map/Software Update, d) Other, and e) Mix workloads.

latencies in the overall latency for the proposed framework. The figure shows the dominance of the communication latency contributing to the overall latency with 46% to 82%. We observe a relatively higher share of the communication latency for the high-priority services, compared to the normal services (82% vs 73%) in the Mix workload.

Fig. 8 shows the number of accesses to the storage subsystem, as a metric demonstrating the efficiency of I/O cache mechanism. The figure shows that the proposal reduces the number of accesses to the storage subsystem by up to 25x. In Collision and Other workloads, we have services with random access pattern. Hence, an efficient I/O cache resource management can reduce the number of storage subsystem accesses, which describes the relatively lower number of storage subsystem accesses in the proposed method, PADSA, and ETICA compared to WSVN. Sensing and Update workloads, however, have sequential access patterns with no temporal locality of accesses. Hence, the I/O cache cannot reduce the number of storage subsystem accesses. As a result, all examined methods have similar number of storage subsystem accesses in the Sensing and Update workloads. The Mix workload is the most challenging in terms of resources management, as it has a combination of high-priority and normal-priority services. We observe that the proposed method and PADSA perform better than ETICA, as ETICA does not consider the service priority in allocating I/O cache resources.

Fig. 9 compares the cache hit ratio of the proposal with the state-of-the-art works. The higher cache hit ratio results in a lower average storage latency. As the figure shows, the proposal outperforms all state-of-the-art-works. While PADSA and ETICA reaches similar cache hit ratio to our proposal (improvement by our proposal in order of few percent), WSVN leads to very low cache hit ratio in most of the workloads except the workloads *b* and *c* representing sensing data upload and map/software update, which are sequential write and sequential read, respectively, and do not benefit from the I/O cache memory. The figure shows that the proposal increases the average cache hit ratio by up to 4.4x.

Fig. 10 shows the impact of monitoring period on the average latency for monitoring period Γ 1 ms to 100,000 ms. If the monitoring period is too large, major changes in the workload behavior can take place without updating the allocated resources, resulting in an inefficient resource usage. The reason is that when the monitoring period is too large, the vehicle moves too far from the associated base station during the monitoring period, resulting in a low transmission rate. Fig. 10 shows a significant increase in the average latency when the monitoring period is beyond 1000 ms, while we do not observe a meaningful improvement in latency when monitoring period is lower than 100 ms. The results also show 24% share of storage in total latency for the Mix workload for monitoring period of 100 ms. The share of storage latency does not noticeably change when decreasing the monitoring period to 1 ms. However, when increasing the monitoring period to 10000 and 100000 ms, the share of storage latency in total latency decreases to 22% and 7%, respectively. Similarly in the rest of workloads, when decreasing the monitoring period to 1 ms, we observe no notable change in the share of the storage latency in the total latency. However, when increasing the monitoring period to 100000 ms, the share of the storage latency in the total latency decreases to 13%, 12%, 10%, and 14% for Collision avoidance, Sensing data upload, Map/software download, and Other workload, respectively.

To demonstrate the scalability of our proposal, Fig. 11 shows

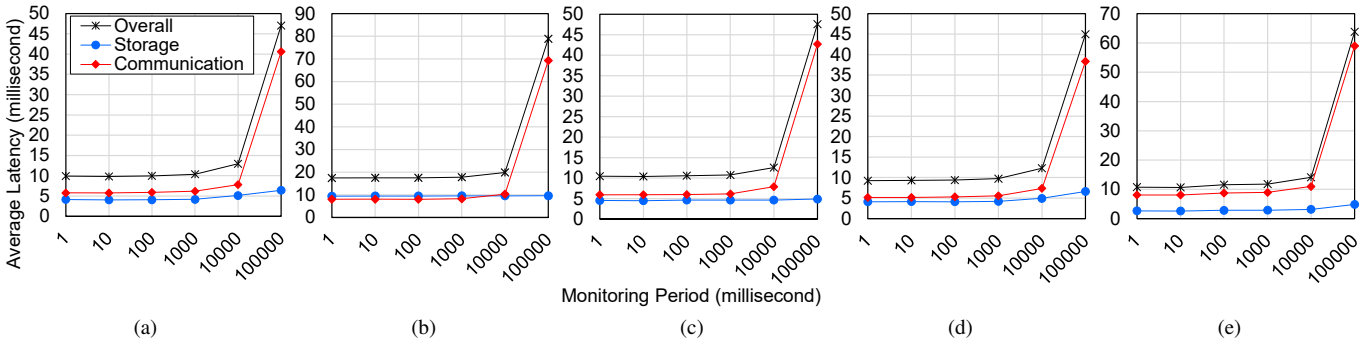

Fig. 10: Impact of monitoring period Γ on performance for: a) Collision Avoidance, b) Sensing Data Upload, c) Map/Software Update, d) Other, and e) Mix workloads.

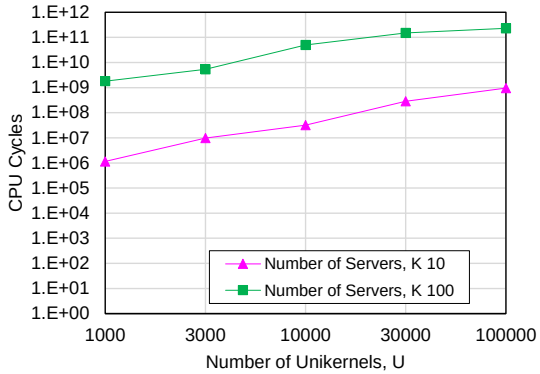

Fig. 11: Execution time of the proposed algorithm for different number of unikernels and edge computing servers.

the execution time of the proposed algorithm for different number of unikernels and edge computing servers in terms of the number of CPU cycles. As the figure shows, when increasing the number of unikernels by 10X (from 1000 to 10,000), the execution cycles increase almost linearly (from 11,500,000 to 318,000,000). By increasing the number of edge computing servers from 10 to 100, however, the execution cycles increase by three orders of magnitude, from 11,500,000 to 18,018,900,000. Finally, by increasing the number of unikernels to 100,000, however, the execution cycles increase to 2.3×10^{12}. We need to mention that the execution cycles is calculated by a single-thread implementation of the algorithm on a single machine, while the potential performance improvements using parallelism can be investigated in our future works.

Fig. 12 shows the performance gap between the optimal solution, the proposed solution, and the state of the art works, for the number of unikernels ranging from 4 to 32. We observe the largest gap between the optimal and proposed solutions for U=4 (11%), while for U=8, U=16, and U=32, the gap is 9%, 8%, and 9%, respectively. The storage latency gap between the proposed and optimal solutions is 10%, 7%, 4%, and 4% for U=4, U=8, U=16, and U=32, respectively. The large latency gap for U=4 is due to the fact that the I/O cache hit ratio is high when U=4 (91% and 90% in the optimal and proposed solution, respectively), as the I/O cache is shared with relatively low number of unikernels. Hence, the average storage latency is relatively low (0.92 ms and 1.02 ms for the optimum and proposed solutions, respectively). For such low storage latency range, even a small improvement in the absolute cache hit ratio results in a relatively large latency improvement due to a notable gap between the main storage and I/O cache latency. As the figure shows, the gap between the related state of the art works and the optimum is way much higher and goes even to hundreds of percent.

VI. Conclusion

In this paper, we have proposed a novel solution for allocation of communication and I/O cache storage resources in the VEC platforms. First, we provide a sub-optimal solution using dual relaxation and we propose two algorithms for a placement of the unikernels to the edge computing servers and we find the optimum I/O cache size in the relaxed problem. To minimize communication latency, we turn the primary assignment problem in an unbalanced bipartite graph into a balanced assignment problem solvable in polynomial time. Finally, we propose a direct heuristic to remove the cache size constraint violation in some edge computing servers. The proposed framework reduces the average latency of all services and the average latency of high-priority services by up to 1.8x and 2.5x, respectively, compared to the state-of-the-art works.

In the future work, the impact of computing latency should be taken into account for optimization.

References

[1] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] Y. Lu, J. Zhang, Y. Qi, S. Qi, Y. Zheng, Y. Liu, H. Song, and W. Wei, "Accelerating at the edge: A storage-elastic blockchain for latency-sensitive vehicular edge computing," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 8, pp. 11 862–11 876, 2021.

[4] M. Kishani, Z. Becvar, and H. Asadi, "Padsa: Priority-aware block data storage architecture for edge cloud serving autonomous vehicles," in *IEEE VNC*, 2021.

[5] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

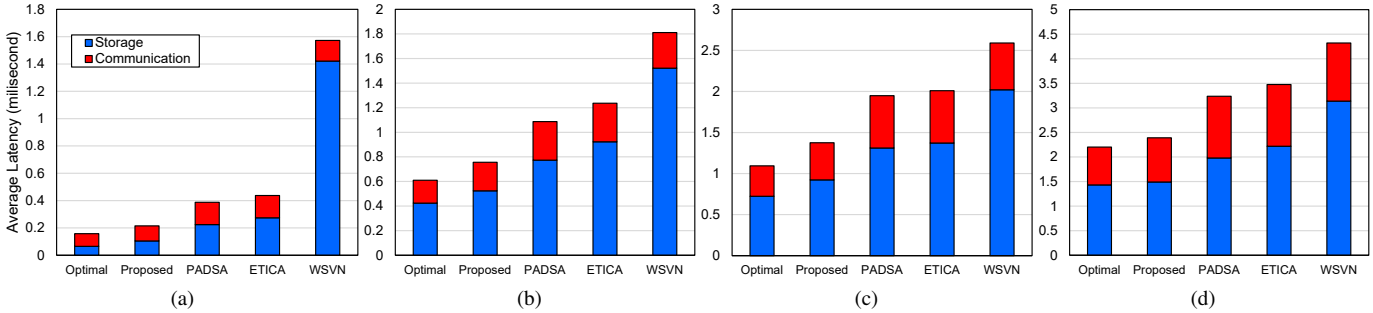

Fig. 12: Performance comparison between the proposed method, previous works, and the optimal solution for: a) $U = 4$, b) $U = 8$, c) $U = 16$, and d) $U = 32$, considering the Mix workload and four base stations ($K = 4$).

[6] W. Balid, H. Tafish, and H. H. Refai, "Intelligent vehicle counting and classification sensor for real-time traffic surveillance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1784–1794, 2017.

[7] S. Amini, I. Gerostathopoulos, and C. Prehofer, "Big data analytics architecture for real-time traffic control," in *2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS)*. IEEE, 2017, pp. 710–715.

[8] J. Yu, H. Zhu, H. Han, Y. J. Chen, J. Yang, Y. Zhu, Z. Chen, G. Xue, and M. Li, "Senspeed: Sensing driving conditions to estimate vehicle speed in urban environments," *IEEE Transactions on Mobile Computing*, vol. 15, no. 1, pp. 202–216, 2015.

[9] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[10] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[11] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile networks and applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[12] X. Xu, Z. Fang, J. Zhang, Q. He, D. Yu, L. Qi, and W. Dou, "Edge content caching with deep spatiotemporal residual network for iov in smart city," *ACM TOSN*, vol. 17, no. 3, pp. 1–33, 2021.

[13] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2017.

[14] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[15] M. M. K. Tareq, O. Semiari, M. A. Salehi, and W. Saad, "Ultra reliable, low latency vehicle-to-infrastructure wireless communications with edge computing," in *GLOBECOM*. IEEE, 2018.

[16] X. Xu, Y. Xue, L. Qi, Y. Yuan, X. Zhang, T. Umer, and S. Wan, "An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles," *Future Generation Computer Systems*, vol. 96, pp. 89–100, 2019.

[17] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1079–1092, 2018.

[18] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2018.

[19] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, and Y. Liu, "joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 4, pp. 4277–4292, 2023.

[20] Y. Jia, C. Zhang, Y. Huang, and W. Zhang, "Lyapunov optimization based mobile edge computing for internet of vehicles systems," *IEEE Transactions on Communications*, vol. 70, no. 11, pp. 7418–7433, 2022.

[21] C. Ren, G. Zhang, X. Gu, and Y. Li, "Computing offloading in vehicular edge computing networks: Full or partial offloading?" in *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, vol. 6. IEEE, 2022, pp. 693–698.

[22] Y. Liu, J. Zhou, D. Tian, Z. Sheng, X. Duan, G. Qu, and V. C. Leung, "Joint communication and computation resource scheduling of a uav-assisted mobile edge computing system for platooning vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8435–8450, 2021.

[23] X. Han, D. Tian, Z. Sheng, X. Duan, J. Zhou, W. Hao, K. Long, M. Chen, and V. C. Leung, "Reliability-aware joint optimization for cooperative vehicular communication and computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5437–5446, 2020.

[24] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[25] S. Ahmadian, R. Salkhordeh, O. Mutlu, and H. Asadi, "Etica: efficient two-level i/o caching architecture for virtualized platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2415–2433, 2021.

[26] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated server flash cache space management in a virtualization environment," in *USENIX Annual Technical Conference*, 2014.

[27] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "Cloud-cache: On-demand flash cache management for cloud computing," in *File and Storage Technologies (FAST)*, 2016, pp. 355–369.

[28] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-cave: Effective ssd caching to improve virtual machine storage performance," in *International conference on Parallel architectures and compilation techniques*. IEEE, 2013, pp. 103–112.

[29] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-side SSD caching for storage performance control," in *International Conference on Autonomic Computing (ICAC)*, 2015.

[30] ——, "Centaur: Host-side ssd caching for storage performance control," in *IEEE ICAC*. IEEE, 2015.

[31] M. Kishani, Z. Becvar, M. Nikooroo, and H. Asadi, "Reducing storage and communication latencies in vehicular edge cloud," in *EuCNC6G Summit*, 2022.

[32] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, and D. Kutscher, "Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions," in *Workshop on Mobile Edge Communications*, 2017.

[33] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, 2017.

[34] M. Zhao, J.-J. Yu, W.-T. Li, D. Liu, S. Yao, W. Feng, C. She, and T. Q. Quek, "Energy-aware task offloading and resource allocation for time-sensitive services in mobile edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10 925–10 940, 2021.

[35] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing," *IEEE access*, vol. 5, pp. 11 255–11 268, 2017.

[36] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8472–8486, 2019.

[37] N. P. Jouppi, "Cache write policies and performance," *ACM SIGARCH Computer Architecture News*, vol. 21, no. 2, pp. 191–201, 1993.

[38] (2022) Stec enhanceio ssd caching software. [Online]. Available: https://github.com/stec-inc/EnhanceIO
[39] (2022) Open cache acceleration software (cas) linux. [Online]. Available: https://github.com/Open-CAS/open-cas-linux
[40] (2022) Linux device mapper cache. [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/cache.html
[41] (2023) Leaderboard – best hard drive, ssd and storage solutions. [Online]. Available: https://www.storagereview.com/best_drives
[42] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds," in *MASCOTS*. IEEE, 2011, pp. 227–236.
[43] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 404–418, 2001.
[44] C. A. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient MRC construction with SHARDS," in *File and Storage Technologies (FAST)*, 2015.
[45] N. Beckmann, H. Chen, and A. Cidon, "{LHD}: Improving cache hit rate by maximizing hit density," in *NSDI*, 2018.
[46] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: Performance insulation for shared storage servers." in *FAST*, vol. 7, 2007, pp. 5–5.
[47] M. Saxena, M. M. Swift, and Y. Zhang, "Flashtier: a lightweight, consistent and durable storage cache," in *ACM european conference on Computer Systems*, 2012, pp. 267–280.
[48] J. Coffey, "Latency in optical fiber systems," COMMSCOPE, Tech. Rep., 01 2023. [Online]. Available: https://www.commscope.com/globalassets/digizuite/2799-latency-in-optical-fiber-systems-wp-111432-en.pdf
[49] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems," in *ITSC*, 2018.
[50] (2021) Fio: Flexible I/O tester synthetic benchmark. [Online]. Available: https://github.com/axboe/fio
[51] (2021) blktrace: A block layer I/O tracing tool. [Online]. Available: https://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html

APPENDIX

In this Appendix, we provide a proof of Proposition 2.

Proof. We use mathematical induction. For the initial step ($i = 1$), the optimal solution has a non-zero number of cache chunks assigned to the unikernel with the maximum objective achievement α from the edge computing server e which base station has the highest channel quality. Proof by contradiction. Suppose the unikernel r has the maximum objective achievement and zero chunks are assigned to the unikernel r from the edge computing server e in the optimal solution m. Furthermore, suppose the unikernel j ($j \neq r$) has at least one cache chunk from the server e in the solution m. In the solution m, we get one cache chunk from j and assign it to r, calling it m'. The objective metric in the solution m is:

$$\begin{aligned} m = \sum_{k=1}^{N} \sum_{i=1,(i,k)\notin\{(r,e),(j,e)\}}^{N_k} \rho_{v,k_i} \times F_{v,k_i} \times D_{v,k_i} \\ +\rho_{j,e} \times F_{j,e} \times (\eta_{j,e} \times L_c + (1-\eta_{j,e}) \times L_m + D^C_{j,e}) \\ +\rho_{r,e} \times F_{r,e} \times (\eta_{r,e} \times L_c + (1-\eta_{r,e}) \times L_m + D^C_{r,e}) \end{aligned} \tag{15}$$

Regarding (14) and (15), the objective metric in m' is:

$$\begin{aligned} m' = m - (L_m - L_c) \times \\ (\rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q_{r,e}) - \rho_{j,e} \times F_{j,e} \times \Psi_{j,e}(Q_{j,e}-1)) \end{aligned} \tag{16}$$

In (16), $(L_m - L_c)$ is always positive regarding (5). Besides, based on our assumption, unikernel r has the maximum objective achievement α. Hence, considering $Q_{r,e} = 0$, we have:

$$\rho_{j,e} \times F_{j,e} \times \Psi_{j,e}(Q_{j,e}-1) < \rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(0) \tag{17}$$

The second factor of A is also always positive. Accordingly, $m' < m$ that contradicts the first assumption that m is the optimum solution.

Next, assume that i cache chunks are assigned to the unikernels following the proposed algorithm and all allocated cache chunks appear in the optimal solution. Now, we prove this assumption for the $(i+1)$-th cache chunk assignment. Suppose the unikernel r has the maximum objective achievement in the current allocation of i cache chunks, while the number of cache chunks already allocated to r is $Q_{r,e}$:

$$\begin{aligned} \rho_{j,k} \times F_{j,k} \times \Psi_{j,k}(Q_{j,k}) \leq \rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q_{r,e}), \\ \forall k \in \langle 1, K \rangle, \forall j \in \langle 1, N_k \rangle \end{aligned} \tag{18}$$

Claim: The optimal solution m allocates at least $Q^m_{r,e} = Q_{r,e} + 1$ cache chunks to unikernel r, hence, $Q_{r,e} < Q^m_{r,e}$, where $Q^m_{r,e}$ is the number of cache chunks allocated to the unikernel r in the optimal solution m.

Proof: Using contradiction, suppose that $Q^m_{r,e} \leq Q_{r,e}$. Based on our assumption, m contains all first i chunks assigned using the firsts i iterations of the while loop (line 4) of Algorithm 2. Hence, for all unikernel, the amount of the allocated cache in m is greater or equal than that of the iteration i of the while loop of Algorithm 2:

$$Q_{j,k} \leq Q^m_{j,k}, \forall k \in \langle 1, K \rangle, \forall j \in \langle 1, N_k \rangle \tag{19}$$

From (19) we have $Q_{r,e} \leq Q^m_{r,e}$, while from our first assumption in the contradiction we have $Q^m_{r,e} \leq Q_{r,e}$. Hence, $Q_{r,e} = Q^m_{r,e}$. It means that no more cache chunks will be allocated to unikernel r after the iteration i of Algorithm 2. So exists unikernel j from edge computing server k, $(j,k) \neq (r,e)$, which cache size in the optimal solution m, $Q^m_{j,k}$, is greater than its size in the iteration i, $Q_{j,k}$:

$$\exists k \in \langle 1, K \rangle, \exists j \in \langle 1, N_k \rangle, (j,k) \neq (r,e), Q_{j,k} < Q^m_{j,k} \tag{20}$$

We define a feasible solution m' as follows; In the optimal solution m, we take one cache chunk from the unikernel j and assign the cache chunk to the unikernel r, naming it solution m'. Solution m' is feasible, as it does not change the total number of allocated cache resources. From (16), we have:

$$\begin{aligned} m' = m - (L_m - L_c) \times \\ (\rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q^m_{r,e}) - \rho_{j,k} \times F_{j,k} \times \Psi_{j,k}(Q^m_{j,k}-1)) \end{aligned} \tag{21}$$

In the following, we show that in (21), m is subtracted by a non-negative value. In (21), $(L_m - L_c)$ is always positive, based on our assumption in (5). Based on our system model assumption (Section II), the hit ratio η is a concave function and Ψ is a monotonically non-increasing function of Q. Hence:

$$\Psi_{j,k}(Q_{j,k}+1) \leq \Psi_{j,k}(Q_{j,k}), \forall k \in \langle 1, K \rangle, \forall j \in \langle 1, N_k \rangle \tag{22}$$

From (20), we get $Q_{j,k} \leq Q^m_{j,k} - 1$. Afterwards, from (18) and (22), we obtain:

$$\begin{aligned} \rho_{j,k} \times F_{j,k} \times \Psi_{j,k}(Q^m_{j,k}-1) \leq \rho_{j,k} \times F_{j,k} \times \Psi_{j,k}(Q_{j,k}) \\ \leq \rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q_{r,e}) = \rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q^m_{r,e}) \end{aligned} \tag{23}$$

Hence:

$$0 \leq \rho_{r,e} \times F_{r,e} \times \Psi_{r,e}(Q^m_{r,e}) - \rho_{j,k} \times F_{j,k} \times \Psi_{j,k}(Q^m_{j,k} - 1) \quad (24)$$

which concludes that in (21), m is subtracted by a non-negative value. Hence, $m' \leq m$. Based on our proof assumption, m is the optimal solution resulting to the minimum objective metric. Hence, $m \leq m'$ and in conclusion, $m = m'$. Hence, m is the optimal solution while it allocates $Q_{r,e} + 1$ cache chunks to unikernel r. Hence, $Q_{r,e} < Q^m_{r,e}$ which contradicts our assumption that $Q^m_{r,e} \leq Q_{r,e}$. Hence the claim is proved and the optimal solution m allocates at least $Q^m_{r,e} = Q_{r,e} + 1$ cache chunks to the unikernel r. As in the proposed algorithm the $i+1$-th cache chunk is allocated to the unikernel r, we conclude that following the proposed algorithm, all $i+1$ cache chunks assigned to the unikernels appear in the optimal solution, and the mathematical induction is proved for $i+1$. Hence, the cache chunk assignment following Algorithm 2 is optimal.

□

Mostafa Kishani received the B.S. degree in computer engineering from Ferdowsi University of Mashhad, Mashhad, Iran, in 2008, M.S. degree in computer Engineering from Amirkabir University of Technology (AUT), Tehran, Iran, in 2010, and PhD degree in computer engineering from Sharif University of Technology (SUT), Tehran, Iran, in 2018. He is currently a research scientist in the Czech Technical University in Prague, Czech Republic. Previously he was a postdoctoral researcher in computer engineering department of SUT. From September 2015 to April 2016 he was a research assistant in Computer Science and Engineering department of the Chinese University of Hong Kong (CUHK), Hong Kong. He was also a research associate in the Hong Kong Polytechnic University (PolyU), Hong Kong, from April 2016 to February 2017.

Mohammadsaleh Nikooroo received B.Sc. degree in Electrical Engineering from Sharif University of Technology, Iran in 2014, and MPhil degree in Information Engineering from the Chinese University of Hong Kong, Hong Kong in 2017. Currently, he is a PhD student at the department of Telecommunication Engineering at Czech Technical University in Prague, Czech Republic. His research project concerns communications in self-optimizing mobile networks with drones with a focus on energy-consumption, communication scheduling, and user's QoS aspects. His research interests include mobile communications, signal processing, channel coding, and machine learning.

Zdenek Becvar (Senior Member, IEEE) received M.Sc. and Ph.D. in Telecommunication Engineering from the Czech Technical University in Prague, Czech Republic in 2005 and 2010, respectively. From 2006 to 2007, he joined the Sitronics R&D center in Prague focusing on speech quality in VoIP. Furthermore, he was involved in research activities of the Vodafone R&D center at the Czech Technical University in Prague in 2009. He was on internships at Budapest Politechnic, Hungary (2007), CEA-Leti, France (2013), and EURECOM, France (2016, 2019). From 2013 to 2017, he was a representative of the Czech Technical University in Prague in ETSI and 3GPP standardization organizations. Now, he is Associate Professor at the Department of Telecommunication Engineering, Czech Technical University in Prague and he leads 6Gmobile research lab at the same university. He has published four book chapters and more than 100 conference or journal papers. His research is focused on development of solutions for future mobile networks with special focus on optimization of mobility and radio resource management, energy efficiency, device-to-device communication, edge computing, C-RAN, self-optimization, and UAVs/vehicular/IoT communications.

Hossein Asadi (M'08, SM'14) received the BSc and MSc degrees in computer engineering from the SUT, Tehran, Iran, in 2000 and 2002, respectively, and the PhD degree in computer engineering from Northeastern University, Boston, MA, USA, in 2007. He is currently a full professor in Department of Computer engineering at SUT. He is the founder and director of the *Data Storage, Networks, and Processing* (DSN) Laboratory and the director of Sharif HPC Center. Dr. Asadi was a recipient of the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, the Distinguished Technology Award from SUT in 2017, and the Distinguished Research Lab Award from SUT in 2019. He also received the Best Paper Award at IEEE/ACM Design, Automation, and Test in Europe (DATE) in 2019. More recently, he received Distinguished National Technology Award in 2022 by Ministry of Science & Technology and Distinguished Research Award in 2022 by SUT. His current research interests include data storage systems, SSDs, operating systems, and high-performance computing.