Reducing Computation, Communication, and Storage Latency in Vehicular Edge Computing

Mostafa Kishani, Zdenek Becvar
[1]*Dpt. of Telecommunication Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague*
Prague, Czech Republic
kishamos@fel.cvut.cz, zdenek.becvar@fel.cvut.cz

***Abstract*—This paper addresses the challenge of optimizing communication, computation, and storage I/O caching in Vehicular Edge Computing (VEC) platforms for autonomous vehicles. The exponential data generated by the autonomous vehicles demands low-latency connectivity with nearby edge servers. However, the existing VEC platforms struggle to meet the performance requirements, especially in real-time applications like collision avoidance. This work proposes a novel algorithm for joint allocation of computing resources, storage I/O cache, and communication resources, considering the diverse priorities and demands of key vehicular services. Our approach integrates application-specific optimizations, prioritization, and joint latency reduction considering communication, computation, as well as storage. Accounting for distinct priorities and data access characteristics of various vehicular services, our proposed feasible solution, employing dual decomposition and Lagrangian relaxation, significantly reduces service latency by up to 64% compared to the current state-of-the-art resource allocation in vehicular edge computing.**

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

I. Introduction

Autonomous vehicles necessitate an exchange of a substantial volume of data with a communication network infrastructure. To ensure prompt delivery of time-sensitive vehicular services, it is imperative to employ low-latency communication, processing, and data access/storage of vehicular services to edge computing servers situated in a close proximity to the vehicles [1]–[3].

The performance of storage poses a significant bottleneck in VEC, as current VEC platforms struggle to meet the performance requirements for applications in autonomous vehicles. Autonomous vehicles generate hundreds of gigabytes of data per vehicle per day, necessitating real-time connectivity with a very low latency [1]. Meeting the demands of real-time collision avoidance, continuous monitoring, or infotainment applications in the autonomous vehicles requires a robust data access and storage in the VEC, i.e., in the areas where existing VEC platforms fall short [1], [2]. Constraints of the autonomous vehicles also highlight computation and storage as major bottlenecks in applications like localization, object detection, or object tracking [3].

In VEC, allocating computation, storage, and communication resources to vehicular services with varying priorities poses a challenge. Safety- and time-critical services, such as collision avoidance, take precedence over other services like software updates [1]. Therefore, the request handling latency for high-priority services should be considered, emphasizing the need for prioritization. The VEC platforms share the storage, computation, and communication resources among different vehicular applications, irrespective of priority and storage/computation/communication demands. Inefficient resource management can impact the performance of critical applications due to the influence of data/processing-intensive applications on overall performance. The request handling latency in VEC is dependent on resources allocated for computation, storage Input/Output (I/O) caching, and communication. Joint allocation of computation, storage I/O caching, and communication resources is crucial, but solving this optimization problem is complex, if not impossible, under realistic assumptions [4].

Previous works on latency reduction in VEC mainly focus on computation, communication, and network caching, overlooking the access to storage. In a similar way, also works on computation offloading often consider CPU cycles, but neglect the access to storage, which is, however, a significant source of latency [5]–[7]. Only two recent studies [2], [8] consider the storage I/O cache latency alongside communication latency in VEC. However, these works ignore the computation latency.

Data storage systems predominantly rely on Hard Disk Drives (HDDs) and low-end Solid-State Drives (SSDs), which have relatively high latency. To achieve a desired performance with low latency, I/O caching is widely used. An efficient allocation of the I/O cache resources is crucial for reducing storage latency in edge and cloud computing systems [9], [10]. However, existing works in this domain often neglect application priority, focus on general data center applications, and do not consider communication and computation aspects. Priority considerations, application-specific optimizations, and joint optimization of communication, computation, and storage latencies are critical for the VEC, but these aspects are not addressed in previous research works.

This paper focuses on the challenge of joint optimization of communication, computation, and storage I/O caching re-

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 No. LTT20004.

source allocation in VEC platforms for key vehicular services, such as collision avoidance, map/software download, and sensing data upload, with different priorities and computation/storage/communication demands. The major contributions of this paper are summarized as follows.

- We propose an algorithm determining the placement of computing resources (in a form of the unikernels) for individual services and allocate the communication and computing resources and I/O cache.
- To attain a computationally viable algorithm for the resource allocation, we employ dual decomposition and Lagrangian relaxation.
- We demonstrate a significant (up to 64%) reduction in the latency of services compared to the current state-of-the-art resource allocation in VEC in realistic scenarios.

The rest of paper is organized as follows. Section II introduces the system model. Section III formulates the problem tackled in this paper. The proposed solution is detailed in Section IV. Following that, Section V outlines the setup for performance evaluation and discusses simulation results. Lastly, Section VI provides the concluding remarks for the paper.

II. System Model

In this section, we present the system model for VEC. After presenting the high-level overview of the system, we elaborate models of the communication, computation, and storage subsystems.

A. System Model overview

Fig. 1 shows the overview of the system model. We assume K VEC servers, individually collocated with K base stations serving V vehicles. Each vehicle may require one or multiple vehicular services, such as collision avoidance or sensing data upload, and each vehicular service is handled by an independent unikernel in the VEC server [11]. Like in typical practical applications, we assume each unikernel is hosted at just one server. We use the notation $U_{v,i}$ for the unikernel i which serves the vehicle v. To simplify the notation, but without any impact on the proposed solution itself, we use U_i to denote $U_{v,i}$, since we assume that all unikernels have a unique index across all vehicles. The unikernel to server placement is determined by the association matrix $A = \{a_{u,k}\}$, $a_{u,k} \in \{0,1\}$ and $a_{u,k} = 1$ indicates the association of the u-th ($u \in \langle 1, U \rangle$) unikernel to the k-th ($k \in \langle 1, K \rangle$) server. Considering N_k as the number of unikernels hosted by the server k, $N_k = \sum_{i=1}^{U} a_{i,k}, \forall k \in \langle 1, K \rangle$.

A hypervisor is responsible for allocation of the communication, computation, and storage resources of the base station and its collocated server to the hosted unikernels regarding parameters such as unikernel priority, number of requests, resource demands of unikernel requests, and the data access pattern. The resource allocation is updated at regular intervals, defined by Γ, while the efficient Γ depends on the mobility/speed, as well as the rate of changes in the vehicular service behavior. The smaller Γ is preferred when the service behavior changes so fast or the vehicles move with high speed.

We assume each request issued by the vehicle is defined in terms of communications, computing, and storage demands and each of these corresponds to specific latency imposed by communication, computing, and storage access, which in turn is a function of resources allocated to address the request. The overall latency is calculated as the aggregation of communication, computation, and storage latency, as shown in Fig. 1. Each VEC request is defined by the following characteristics:

- **Communication demand:** Includes uplink request with size l_{UL} bits, sent from a vehicle to a base station, requesting a specific service, and downlink response with size l_{DL} bits, sent from the base station to the vehicle.
- **Computation demand:** In terms of the number of Floating-Point Operations (FLOP) l_{CP} needed to accomplish the computation.
- **Storage demand:** Includes storage read data size l_R and storage write data size l_W in bits.
- **Priority** A predefined value indicating the priority of the request. This value, indicated by $\rho \in \langle 0, 1 \rangle$, is usually a function of the service type.

Each vehicular request, corresponding to a specific vehicular service, carries the identifier of unikernel responsible for handling the service. Hence, each request received in the base station is handled by the specified unikernel. Each unikernel handles and responds its received requests independently, using the allocated resources. The base station allocates the communication, computation, and storage resources to individual unikernels, aiming to minimize/maximize an objective metric which is a combination of the following criteria:

1) Average latency per request: Average latency D_{v,k_i} per request is the average time from issuing the request by the vehicle to receiving the response, calculated as the aggregation of average communication latency, average computation latency, and average storage latency.

2) Average priority per request: $\rho_i \in \langle 0, 1 \rangle$ indicates the average priority of requests handled by the unikernel i.

3) Number of requests: The number of requests per unit of time handled by the unikernel i.

B. Communication Latency

Both uplink and downlink communication latency is considered in our system model.

1) Uplink Latency: Uplink latency is the latency of uploading l_{UL} bits of data from the vehicle v to the base station k hosting the unikernel i:

$$D_{v,k_i}^{UL} = \frac{l_{UL}}{R_{v,k_i}^{UL}} \tag{1}$$

where D_{v,k_i}^{UL} is uplink latency, l_{UL} is uplink demand, and R_{v,k_i}^{UL} is uplink data rate at the radio channel between the vehicle v and the base station:

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

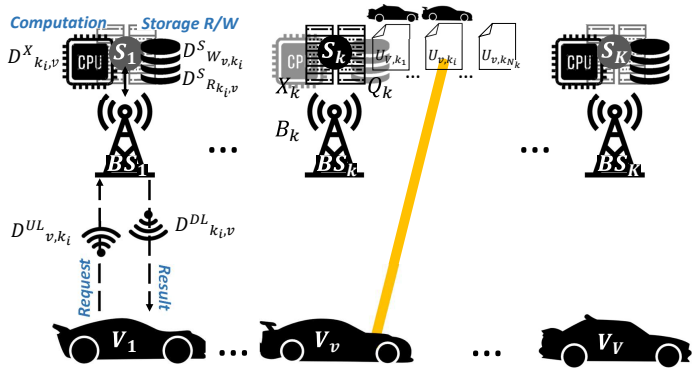

Fig. 1. High-level overview of the system model. Total number of V vehicles are served by K base stations, while base station k collocates a single server hosting N_k number of unikernels. Each unikernel is responsible to handle a single service of a single vehicle.

where B_{v,k_i} is the communication bandwidth allocated to the channel between the vehicle and the base station, P^R_{v,k_i} is the power of received signal at the base station, σ is the thermal noise, and I_k is the interference coming from all concurrent communications with base stations other than the base station k.

2) Downlink Latency: Downlink latency, is the latency of downloading l_{DL} bits of data from base station k hosting unikernel i to vehicle v:

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

where $D^{DL}_{k_i,v}$ is downlink latency, l_{DL} is downlink demand, and $R^{DL}_{k_i,v}$ is the downlink data rate of radio channel between the base station k hosting the unikernel i and vehicle v:

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

where $B_{k_i,v}$ is the communication bandwidth between the base station and the vehicle, $P^R_{V_{k_i,v}}$ is the power of received signal at the vehicle, σ is the thermal noise, and $I_{k,v}$ is the interference of other base stations, except the base station k. In each base station, we assume that the communication bandwidth and transmission power is proportionally divided by the number of hosted unikernels. Hence, considering B^{BS}_k as the bandwidth of base station k and $P^{T_{BS}}_k$ as the transmission power of base station k, we have $B_{k_i,v} = \frac{B^{BS}_k}{N_k}$ and $P^T_{k_i,v} = \frac{P^{T_{BS}}_k}{N_k}$, where $P^T_{k_i,v}$ is the transmission power allocated to U_i by the base station k. Similarly in each vehicle, considering B^V_v as the communication bandwidth of vehicle v, $P^{T_V}_v$ as the transmission power of vehicle v, and N^V_v as the number of unikernels per vehicle, we assume $B_{v,k_i} = \frac{B^V_v}{N^V_v}$ and $P^T_{v,k_i} = \frac{P^{T_V}_v}{N^V_v}$, where B_{v,k_i} is bandwidth of vehicle-base station communication and P^T_{v,k_i} is the transmission power allocated to U_i by vehicle v. The communication bandwidth that the vehicle and base station allocate should be identical. However, due to the unpredictability and resource restrictions in the base station side, we assume that only the base station side has a bandwidth constraint for both uplink and downlink communications. Hence, $B_{k_i,v} = B_{v,k_i} = \frac{B_k}{N_k}$. As each unikernel has a unique index $i \in \langle 1, U \rangle$, for the sake of brevity, hereafter we respectively use B_i and P_i notations to refer $B_{k_i,v}$ and $P^T_{k_i,v}$.

C. Computation Latency

The computation latency, $D^{CP}_{k_i,v}$, is the computation latency of handling the vehicular request by the VEC server, issued from the vehicle v to the server k hosting the unikernel i. $D^{CP}_{k_i,v}$ is a function of computation demand, l_{CP}, the number of processing elements (processor cores) allocated to the unikernel, X_i, and the processing power of each processing element in terms of Floating-point Operations Per Second (FLOPS), χ. We assume the computation can be parallelized over any arbitrary number of available processing elements. We also assume the parallelism overhead inside a VEC server, including the overhead of inter-processor communication and the aggregation of final result, is negligible compared to the total computation and communication time [7]. Hence, we assume that the processing time linearly decreases with increasing the level of parallelism [7].

$$D^{CP}_{k_i,v} = \frac{l_{CP}}{\chi \times X_i} \tag{5}$$

We assume each unikernel only uses the local processing elements of the host server.

D. Storage Latency

We assume each unikernel only uses the local I/O cache and storage resources of the host server. We also assume that the installed storage capacity on each server is always greater than the needs of hosting unikernels. This is a practical assumption, due to the relative affordability of the main storage media, enabling the system designers to provision a capacity larger than the demands of normal missions. We assume the *Solid State Drive* (SSD) for the main storage media and high-end *Non-Volatile Memory* (NVM) for the I/O cache. We assume conventional best practices for the I/O cache, including the write-back policy and *Least Recently Used* (LRU) replacement policy [12], [13]. We also assume that the missed read requests are promoted from the main storage to the I/O cache [14]. Considering the technical specification of commercial SSDs and NVMs, NVMs perform better than SSDs in terms of both read/write latency and *Input/Output Per Second* (IOPS). Hence, considering L_m and L_c respectively as the latency of main storage and I/O cache memory:

$$L_c < L_m \tag{6}$$

1) Modeling Storage Latency: A storage access is either write or read. As in the write-back policy, write requests are written directly to the I/O cache, they have a similar latency of $D^S_{W_{v,k_i}}$, less influenced by the efficiency of I/O cache management policy [9], [10]. However, the average latency of read requests in each unikernel, $D^S_{R_{k_i,v}}$, is affected by the capacity of allocated I/O cache, mandating an efficient allocation policy. $D^S_{R_{k_i,v}}$ is a function of I/O cache hit ratio, as well as the latency of I/O cache memory and main storage [12], [15]. I/O cache

hit ratio, in turn, is a function of the allocated I/O cache size. Considering Q_i as the I/O cache size allocated to the unikernel i and $\eta_i(Q_i)$ as the I/O cache hit ratio of U_i:

$$D^S_{k_i,v} = l_R \times (\eta_i(Q_i) \times L_c + (1 - \eta_i(Q_i)) \times L_m) \tag{7}$$

2) Developing Hit Ratio Curve: To develop the hit ratio curve, $\eta_i(Q_i)$, for each unikernel, we use stack distance analysis on the history of data accesses in the previous monitoring period. Per storage access, the stack distance is defined as the number of accesses to distinct storage addresses, after the previous access to the same address. When using LRU replacement policy, an I/O cache size equal or greater than $s+1$ assures a cache hit on all accesses with a stack distance not larger than s. Hence, considering Ψ_i and N^A_i respectively as the stack distance histogram of storage accesses and the total number of storage accesses in U_i:

$$\eta_i(Q_i) = \frac{\sum_{s=0}^{Q_i-1} \Psi_i(s)}{N^A_i} \tag{8}$$

Given Δ_i as the maximum stack distance of Ψ_i, we define the ideal cache size as $\Delta_i + 1$, for which we do not get performance improvement by any larger cache size. We assume that $\eta_i(Q_i)$ is a concave function when LRU replacement policy is used [16]. Hence, $\Psi_i(s)$ is a monotonically non-increasing function.

E. Total Latency

The total latency per VEC request of U_i is the aggregation of uplink communication latency, downlink communication latency, computation latency, storage write latency, and storage read latency:

$$D_{v,k_i} = D^{UL}_{v,k_i} + D^{DL}_{k_i,v} + D^{CP}_{k_i,v} + D^S_{k_i,v} \tag{9}$$

III. Problem Definition

We aim to reduce the total latency of the services requested by vehicles, taking into account communication, computation, and storage factors. The objective metric should reflect the three unikernel characteristics that we focus on, namely, the average priority per request ρ, the number of requests F, and the average latency per request D (see Section II-A). We define the objective metric as the product of these three characteristics and the association matrix A which defines the unikernel to server (base station) allocation, $a \times \rho \times F \times D$. The search space includes the association matrix A and latency D, while priority ρ and number of requests F are control variables. Our search space affecting the latency includes allocated I/O cache size Q_i, number of allocated CPU cores X_i, and the unikernel to base station association $a_{i,k}$ which affects communication bandwidth B_i.

$$\min \sum_{k=1}^{K} \sum_{i=1}^{U} a_{i,k} \times \rho_i \times F_i \times D_{v,k_i}(X_i, B_i, Q_i) \tag{10}$$

$$s.t. \quad \sum_{i=1}^{K} a_{n,i} = 1, \quad \forall n \in \langle 1, U \rangle \quad (a)$$

$$\sum_{i=1}^{U} a_{i,k} \times X_i \leq X^{BS}_k, \quad \forall k \in \langle 1, K \rangle \quad (b)$$

$$\sum_{i=1}^{U} a_{i,k} \times Q_i \leq Q^{BS}_k, \quad \forall k \in \langle 1, K \rangle \quad (c)$$

$$Q_i \leq \Delta_i + 1, \quad \forall i \in \langle 1, U \rangle \quad (d)$$

$$\sum_{i=1}^{U} a_{i,k} \times B_i \leq B^{BS}_k, \quad \forall k \in \langle 1, K \rangle \quad (e)$$

where U is the number of unikernels, B^{BS}_k is the communication bandwidth of the base station k, X^{BS}_k is the number of processing units in the edge computing server k, and Q^{BS}_k is the size of I/O cache memory in the edge computing server k.

Constraint *(a)* in (10) limits the allocation of each unikernel to one and only one base station. We use the constraint *(b)* and *(c)* to respectively ensure that the processing units and cache memory allocated by the edge computing server k does not exceed the number of processing units and cache memory size of the server k. The constraint *(d)* makes sure that the total cache space allocated to the unikernel i is not larger than the ideal cache space determined by the maximum stack distance, Δ_i. Constraint *(e)* ensures that the communication bandwidth allocated by base station k does not exceed the total bandwidth of the base station k. We assume that the system model can satisfy the possible constraints related to the communication power.

IV. Proposed Solution

In this section, we propose a solution that deals with the assignment of unikernels to base stations, the allocation of computational resources, and the distribution of storage I/O cache.

A. Unikernel to Base Station Association

This section describes a problem of finding a placement of unikernels to edge computing servers that are collocated with base stations. The goal is to minimize the communication objective metric, by relaxing the computation and storage constraints. The communication resources are managed by the base station collocating the edge computing server. Moreover, regarding the system model, each base station collocates one and only one server. Hence, the unikernel to edge computing server association is determined by the unikernel to base station association. Assuming proportional BW distribution over all unikernels associated to a base station, we remove the

communication bandwidth constraint and find the sub-optimal unikernel to base station association, formulated as follows:

$$\min \sum_{k=1}^{K}\sum_{i=1}^{U} a_{i,k} \times \rho_i \times F_i \Big(\frac{l_{UL_i}}{\frac{B_k^{BS}}{\sum_{j=1}^{U} a_{j,k}} \times log_2(1+\frac{P^{T_V}(\frac{\lambda}{4\pi d_{i,k}})^2}{\sigma+I_k})} + \frac{l_{DL_i}}{\frac{B_k^{BS}}{\sum_{j=1}^{U} a_{j,k}} \times log_2(1+\frac{P^{T_{BS}}(\frac{\lambda}{4\pi d_{i,k}})^2}{\sigma+I_{k,v_i}})} \Big) \tag{11}$$

The association problem is a weighted bipartite graph matching in the unikernels part, where the minimum sum of weights is desired. To solve this problem, we turn this graph to a new bipartite graph with equal number of vertices in each part. To this end, we assume having U speculative base stations (assuming U is the number of unikernels and each base stations is associated to $\frac{U}{K}$ speculative base stations), where each speculative base station has the bandwidth of $\frac{B^{BS}\times K}{U}$, given B^{BS} is the bandwidth of each base station and K is the total number of base stations. Afterwards, we use the Hungarian algorithm to find the perfect matching of U unikernels to U speculative base stations. This algorithm allocates equal number of unikernels ($\frac{U}{K}$) to each base station.

B. Computation Resource Allocation

After fixing the unikernel to base station association, we locally optimize the computation resource allocation to each unikernel, formulated as follows:

$$\min \sum_{i=1}^{N_k} \rho_i \times F_i \times \frac{l_{CP_i}}{\chi \times X_i}$$
$$s.t. \quad \sum_{i=1}^{N_k} X_i \leq X_k^{BS} \tag{12}$$

We define $W_i = \rho_i \times F_i \times \frac{l_{CP_i}}{\chi}$ and solve the problem using Lagrangian relaxation. Let the Lagrangian function L be:

$$L = \sum_{i=1}^{N_k} \frac{W_i}{X_i} + \Lambda(\sum_{i=1}^{N_k} X_i - X_k^{BS}) \tag{13}$$

By taking the derivative of L we have:

$$\frac{\partial L}{\partial X_i} = -\frac{W_i}{X_i^2} + \Lambda = 0 \tag{14}$$

Solve it for X_i:

$$X_i = \sqrt[2]{\frac{W_i}{\Lambda}} \tag{15}$$

We can use the constraint $\sum_{i=1}^{N_k} X_i = X_k^{BS}$ to solve for Λ:

$$X_k^{BS} = \sum_{i=1}^{N_k} \sqrt[2]{\frac{W_i}{\Lambda}} \tag{16}$$

Solving for Λ, we get:

$$\Lambda = (\frac{\sum_{i=1}^{N_k} \sqrt[2]{W_i}}{X_k^{BS}})^2 \tag{17}$$

Finally, we substitute this value of Λ into the expression for X_i (15) and substitute W_i:

$$X_i = X_k^{BS} \times \frac{\sqrt[2]{\rho_i \times F_i \times l_{CP_i}}}{\sum_{i=1}^{N_k} \sqrt[2]{\rho_i \times F_i \times l_{CP_i}}} \tag{18}$$

C. Storage I/O Cache Resource Allocation

After fixing the unikernel to base station association, we locally optimize the I/O cache resource allocation to each unikernel.

$$\min \sum_{i=1}^{N_k} \rho_i \times F_i \times l_{R_i} \times (L_m + (L_c - L_m) \times \frac{\sum_{s=0}^{Q_i-1} \Psi_i(s)}{N_i^A})$$
$$s.t.: \quad \sum_{i=1}^{N_k} Q_i \leq Q_k^{BS} \quad (a)$$
$$Q_i \leq \Delta_i + 1 \quad \forall i \in \langle 1, N_k \rangle \quad (b)$$

The term $(L_c - L_m)$ is always negative from (6) and Ψ_i is a monotonically non-increasing function from the system model. Hence, a greater Q_i value reduces the objective metric. We use an optimal algorithm proposed by [1] to find Q_i in polynomial complexity, shown in Algorithm 1.

V. Results

This section outlines the simulation setup and assesses the performance of the proposed framework, focusing on the average latency.

A. Simulation Setup

In our simulations we consider the computation demand proportional to the number of bits transmitted/received, as assumed by Liu et. al. [7], in terms of Floating-Point Operations per bit (FLOP/bit). To assess the effectiveness of the proposed framework under authentic storage workloads, we process the real block-layer traces of the edge computing server that serves the autonomous vehicles. Basic services are generated using the Flexible I/O (FIO) benchmarking tool [17], and storage block accesses for each service are collected using the Linux *blktrace* tool [18]. We establish four vehicular services, namely collision avoidance, sensing data upload, map/update download, and other general services. These services are detailed in Table I.

Simulation experiments are conducted using five representative workloads: Collision avoidance (*Collision*), Sensing data upload (*Sensing*), Map and software update (*Update*), Other services for the autonomous vehicles (*Other*), and a mix of all services with equal probability for each of the four basic service types (*Mix*). Vehicular edge computing is simulated with system model parameters summarized in Table II. Each edge computing server is assumed to be collocated with one base station deployed along the road, maintaining an equal inter-site distance of 1000 m. The vehicle trajectories are

Algorithm 1 Local Optimum I/O Cache Allocation

```
for i ← 1 to N_k do
    Q_i = 0
end for
while Σ_{j=1}^{N_k} Q_j ≤ Q_k^{BS} do
    i = max(ρ_r × F_r × l_{R_i} × Ψ_r(Q_r)), r ∈ ⟨1, N_k⟩
    if Q_i < Δ_i + 1 then
        Q_i = Q_i + 1
end while
```

TABLE I
SERVICE CHARACTERISTICS

	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 (ρ)	Real-time (1)	Normal (0.5)	Normal (0.5)	Normal (0.5)
CPU FLOP/bit	10^4	10^3	10^3	10^4

modeled using data from the highD dataset [19], encompassing the entire traffic data (16.5 hours) with a timescale of 1/25 seconds. Communication bandwidth and power are allocated proportionally to the number of unikernels hosted by the edge computing server of the base station serving the vehicle, considering a carrier frequency of 2.6GHz.

Given that no prior research has concurrently addressed communication, computation, and storage latency together, we conduct a comparative analysis with three leading methods. These methods encompass an I/O cache architecture designed for edge and virtualized environments *Efficient Two-level I/O Caching Architecture for Virtualized Platforms* (ETICA) [9], a platform for storage I/O cache and computation resource allocation for Vehicular Edge Computing (VEC) applications, *Joint Optimization of Communication and Storage Latencies for Vehicular Edge Computing* (JOCS) [2], and a workload scheduling platform for VEC that accounts for communication and computation latencies, referred to as *Workload Scheduling in Vehicular Networks With Edge Cloud Capabilities* (WSVN) [20].

ETICA and WSVN frameworks involve a single server, with ETICA omitting consideration of communication latency. For comparative purposes, we assume that ETICA allocates unikernels to the edge computing server based on free cache space, prioritizing the base station with the lowest distance from the vehicle. This allocation process is assumed to occur sequentially, starting from unikernel 1 and concluding with unikernel U. WSVN considers communication latency for request upload and response download, along with computation latency, albeit without addressing storage latency. To facilitate a meaningful performance comparison, we treat high-priority vehicle services, such as collision avoidance, as analogous to WSVN's *Delay-Intolerant Tasks*. Moreover, we align each storage request in our system model with a computation request in WSVN, assuming that both data size (in bits) and computation requirements (CPU cycles) in WSVN are proportional to the storage access size in our system model. We also assume that JOCS and ETICA allocate the computation resources proportional to the storage access size.

B. Results

Figure 2 illustrates a comparison of the average latency between the proposed framework and state-of-the-art approaches. The depicted data includes the average overall latency across all services, encompassing both high-priority and normal services. The proposed framework showcases an improvement in average latency up to 64% across all examined workloads. This enhancement is attributed to efficient allocation of computation resources, storage I/O cache assignment, and service placement, considering service priority and thereby reducing computation, communication, and storage latency. The most significant latency improvement is observed in the Mix workload, justified by its combination of high-priority and normal-priority services, benefiting significantly from the proposed framework's consideration of service priority and computation latency in the objective metric.

In Collision, Sensing, Update, and Other workloads, the proposed method's superiority over ETICA stems from efficient computation and communication resource allocation. Additionally, its advantage over WSVN is attributed to computation, storage and communication resource allocations. In scenarios such as Collision, Sensing, Update, and Other workloads, where only one type of vehicular service is present, the proposed method exhibits performance slightly better than JOCS. This observation is explained by our simulation assumption, which assumes proportional computation resource allocation for both JOCS and ETICA to have a fair comparison, even though these methods do not deal with computation latency originally. Consequently, the request sizes have minimal variation, the proposed method allocates comparable computation resources to all unikernels of the VEC server, resulting in similar computation latency. However, it is important to note that this alignment may not hold true in realistic applications where a mix of different request sizes with diverse computation demands is encountered, as evidenced by our mixed workload results.

In the case of mixed workload, the proposed method demonstrates superiority due to more efficient storage, computation, and communication resource management, irrespective of service priority. In Sensing and Update workloads, WSVN slightly outperforms ETICA, attributed to their sequential access patterns. Sequential workloads, lacking I/O cache benefits, result in similar storage performance for all methods. Workloads like Collision and Other, featuring random read/write access patterns, benefit from efficient I/O cache resource allocation, reducing storage latency. WSVN, with relatively high storage latency, experiences higher overall latency. The proposed method, similar to JOCS and ETICA in storage latency when services have the same priority, outperforms them in Collision and Other workloads due to more effective communication resource allocation, resulting in lower communication latency.

Figure 3 illustrates the contribution of communication, computation, and storage latencies to the overall latency. The figure

TABLE II
SIMULATION PARAMETERS

Notation	Parameter	Value
K	Number of edge computing servers	10
U	Number of unikernels (4 unikernels per vehicle)	1000
L_c	I/O cache memory (SSD) latency	25 us
L_m	Main storage (HDD) 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
χ	FLOPS of each processing element	20×10^9
X^{BS}	Number of processing elements in each server	100

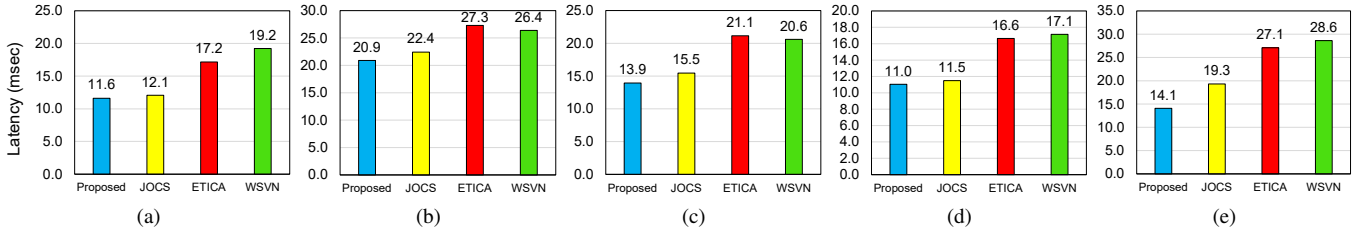

Fig. 2. Aggregation of computation, communication, and storage latency for: a) Collision Avoidance, b) Sensing Data Upload, c) Map/Software Update, d) Other, and e) Mix workloads.

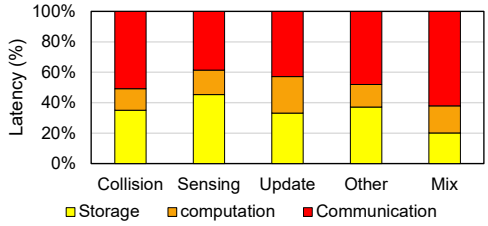

Fig. 3. Share of computation, communication and storage in overall latency.

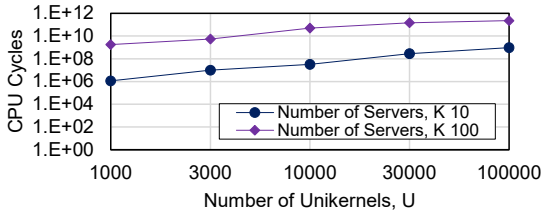

Fig. 4. Running CPU cycles of the proposed algorithm for different number of unikernels and edge computing servers.

highlights the predominant role of communication latency, accounting for 39% to 62% of the overall latency.

To showcase the scalability of our proposed approach, Figure 4 presents the CPU cycle of the algorithm concerning varying numbers of unikernels and edge computing servers. The figure illustrates that augmenting the number of edge computing servers from 10 to 100 results in a three-orders-of-magnitude increase in execution cycles. On the other hand, when the number of unikernels is increased from 1000 to 10,000, the execution cycles exhibit nearly linear growth.

VI. Conclusion

In this paper, we have proposed an innovative approach for the allocation of computation, communication, and storage I/O cache resources within VEC platforms. By prioritizing key vehicular services with diverse demands, we employ dual decomposition and Lagrangian relaxation to achieve a computationally viable solution for unikernel placement, computing resource allocation, and I/O cache resource allocation on the edge computing server. The proposed methodology achieves a remarkable reduction in service latency, demonstrating up to a 64% improvement compared to the prevailing state-of-the-art resource allocation method in VEC.

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] M. Kishani, Z. Becvar, M. Nikooroo, and H. Asadi, "Joint optimization of communication and storage latencies for vehicular edge computing," *IEEE Trans. ITS*, pp. 1–15, 2023.

[3] 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.

[4] 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.

[5] 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 Trans. ITS*, vol. 24, no. 4, pp. 4277–4292, 2023.

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

[7] 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 Trans. ITS*, vol. 23, no. 7, pp. 8435–8450, 2021.

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

[9] 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.

[10] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-side ssd caching for storage performance control," in *IEEE ICAC*. IEEE, 2015.

[11] 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.

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

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

[14] (2024) Linux device mapper cache. [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/cache.html

[15] 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.

[16] N. Beckmann, H. Chen, and A. Cidon, "{LHD}: Improving cache hit rate by maximizing hit density," in *NSDI*, 2018.

[17] (2021) Fio: Flexible I/O tester synthetic benchmark. [Online]. Available: https://github.com/axboe/fio

[18] (2021) blktrace: A block layer I/O tracing tool. [Online]. Available: https://www.cse.unsw.edu.au/ aaronc/iosched/doc/blktrace.html

[19] 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.

[20] 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.