PADSA: Priority-Aware Block Data Storage Architecture for Edge Cloud Serving Autonomous Vehicles

Mostafa Kishani
Dep. of Telecommunication Engineering
Czech Technical University in Prague
Prague, Czech Republic
kishamos@fel.cvut.cz

Zdenek Becvar
Dep. of Telecommunication Engineering
Czech Technical University in Prague
Prague, Czech Republic
zdenek.becvar@fel.cvut.cz

Hossein Asadi
Dep. of Computer Engineering
Sharif University of Technology
Tehran, Iran
asadi@sharif.edu

***Abstract*—An efficient Input/Output (I/O) caching mechanism for data storage can deliver the desired performance at a reasonable cost to edge nodes serving autonomous vehicles. Current storage caching solutions are proposed to address common applications for autonomous vehicles that are less demanding in terms of the latency (e.g., map or software upgrades). However, a serious revision of these solutions is necessary for autonomous vehicles, which rely on safety- and time-critical communication for services, such as collision avoidance, requiring very low latency. In this paper, we propose a three-level storage caching architecture for virtualized edge cloud platforms serving autonomous vehicles. This architecture prioritizes safety-critical services and allocates the two top-level caches of *Dynamic Random Access Memory* (DRAM) and *Non-Volatile Memory* (NVM) to the top priority services. We further evaluate optimum cache space allocated to each service to minimize the average latency. The experimental results show that the proposed architecture reduces the average latency in safety-critical applications by up to 70% compared to the state-of-the-art.**

I. Introduction

Recent advancements in 5G and beyond mobile networks support and enable new applications that were not feasible in previous generations of mobile networks. The emerging application scenarios, such as autonomous vehicles, healthcare, industrial internet, or virtual/augmented reality require high reliability, low latency, and high security/privacy. In specific, the autonomous vehicles are characterized by a huge volume of data transmission exceeding 100 GB per day per vehicle, while each vehicle roughly contains 50-200 IoT-like devices. A part of these transmissions is associated to safety- and time-critical services, such as collision avoidance that mandate a very low latency in order of few milliseconds.

Previous surveys [1–4] assert that 5G networks are supposed to satisfy tight network latency requirements of cloud platforms for autonomous vehicles to provide "beyond the line of sight" visibility to the vehicles. These surveys mention a vehicle to infrastructure communication as a key challenge and propose *Mobile Edge Computing* (MEC) [2] as a suitable solution that can deliver the required low latency, since the MEC hosts the applications close to the users at a low network latency.

The technology stack of autonomous driving includes three major subsystems [5]: i) driving logic subsystem, ii) vehicular edge subsystem, and iii) cloud platform. The driving logic subsystem is responsible for sensing, perception, and decision; the vehicular edge subsystem contains operating system and hardware platforms; and the cloud platform is responsible for data storage, high-definition mapping, and deep learning model training. Existing cloud platforms are mostly responsible for offline computing including, for example, high-definition map production or deep learning model training, while future advancements in communication industry can enable to also handle safety-critical, real-time tasks, such as collision avoidance, by the (edge) cloud [5].

While low-latency and high-bitrate network communication is a major concern in autonomous vehicle applications, a serious performance bottleneck in MEC for the autonomous vehicles is a storage subsystem[1] of the edge nodes. The storage demands of the autonomous vehicles call for a revolution in storage architecture of cloud and edge infrastructures. Services and applications of the autonomous vehicles mostly have a real-time nature, unstructured data type [10, 11], and require very low latency in order of milliseconds. The autonomous vehicles also produce a traffic pattern that is quite different from common internet and smartphone usage. While the majority of present applications have a large downlink traffic and limited uplink traffic, the autonomous vehicles require a heavy and real-time uplink traffic of sensed data while the downlink traffic is usually limited to control commands, maps, and software updates. Due to this big difference between workload characteristics of the autonomous vehicles and common internet applications, the storage stack of edge nodes should be re-architectured. Meantime, while the majority of automotive services mandate very low latency, these services are not on a par with each other. In addition, although we cannot compensate

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.

[1]The storage subsystem is responsible for permanent data storage in a high-reliable and high-available manner [6–9].

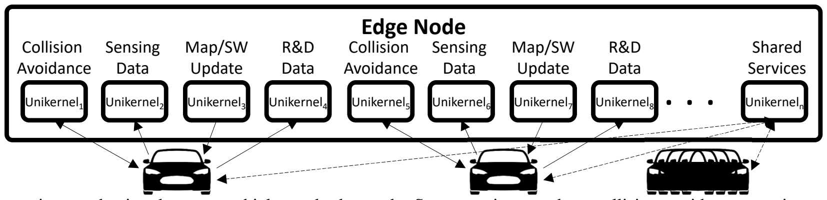


Fig. 1: Overview of service and connection mechanism between vehicles and edge node. Some services, such as collision avoidance, require communication to the services of other vehicles. An alternative is that a single service simultaneously serves multiple vehicles (or all vehicles) in the sight of edge node, shown as *Shared Service* in the figure. Dashed lines represent the communication between shared services and vehicles.

reliability and latency in some safety-critical services (such as collision avoidance) at any cost, some other services, such as software updates, map updates, and data collection for research and development purposes, can tolerate lower performance standards. Hence, to balance latency and storage cost and to prioritize demanding services for the autonomous vehicles, more diverse storage options are required. By developing smart mechanisms such as data tiering and caching, we can dynamically and transparently manage edge storage for a big spectrum of autonomous vehicle services motivating a large investment on emerging storage architectures [5, 11–13].

The edge nodes for the autonomous vehicle services can be located in different facilities, such as base stations (in 5G denoted as gNBs) or *Unmanned Aerial Vehicles* (UAVs) [14–16]. For the sake of security and ease of management and mobility, these edge platforms typically employ virtualization paradigms, such as containers, Virtual Machines (VMs), or Unikernels [13]. In such platforms, the edge node takes over multiple services per vehicle, while each service is associated to an independent unikernel. The unikernel services are typically interconnected and share storage space. There is also a possibility of having shared services among multiple vehicles, or having a single unikernel responsible for multiple related services. Since storage is shared among multiple unikernel services, I/O and storage can become the performance bottleneck of mission-critical services and adversely affect system functionality.

Performance of the storage stack in virtualized platforms have been the subject of many studies [17–21]. However, existing I/O caching methods for virtualized systems do not consider the priority of applications and services [17–21] and most of these methods focus on general datacenter applications and workloads, managing to improve the average latency of all VMs [17, 19–21]. To our best knowledge, there is no previous work addressing the storage subsystem performance in the edge and cloud platforms of the autonomous vehicles. A commonly used approach to manage and improve performance of virtual platforms is I/O caching [17–21]. An efficient I/O caching mechanisms for the autonomous vehicles should: a) consider the worst-case latency of safety- and time-critical services, such as collision avoidance, to assure the safety of passengers, b) prioritize the safety- and time-critical services for limited storage resources, and c) consider realistic autonomous vehicle workloads rather than commodity datacenter applications.

In this paper, we propose a novel priority-aware block data storage architecture for edge cloud serving the autonomous vehicles, denoted as PADSA, represented by a three-level storage caching architecture of Dynamic Random Access Memory (DRAM), Non-Volatile Memory (NVM), and Solid-State Drive (SSD) for virtualized platforms serving autonomous vehicle services. PADSA prioritizes time- and safety-critical services and allocates the two top-level caches of DRAM and NVM to these services. This architecture dynamically manages cache resources allocated to each unikernel during runtime and finds the optimum cache resource allocation regarding the recent workload behavior. Furthermore, we find the optimum cache resource allocation that minimizes the average latency. The efficiency of the proposed architecture is validated using realistic storage traces of the autonomous vehicle services. Our experiment results show that PADSA reduces the average latency of safety-critical applications by up to 70% compared to the state-of-the-art.

The rest of paper is organized as follows. Section II discusses the system model including the edge computing model considered in this work. Section III elaborates the problem and provides its formulation, parameters, and constraints. Section IV presents the proposed priority-aware block data storage architecture. Section V presents the experimental setup and results. Finally, Section VI concludes the paper.

II. System Model

In this section, we outline the edge computing model, including storage stack and virtualization platform addressed in this paper. The edge node serves unikernels from different vehicles, while each unikernel handles an arbitrary service of a specific vehicle. Hence, the edge node hosts multiple unikernels per vehicle for different services, such as collision avoidance, map update, or software update. Fig. 1 illustrates an interconnection between the vehicles and the edge node. The unikernels resemble the VMs while having much lower space and computation complexity, as the unikernel contains only necessary libraries of operating system (OS) rather than the whole OS and has only one addressing space (not switch between user and kernel mode). As the VMs are very heavy to migrate and boot and have large computation overhead due to complex software layers, the more popular solution for the vehicular edge computing are containers running under the management of edge OS, or unikernels running under an edge hypervisor, while the latter solution is getting more popular due to security benefits [13].

The hypervisor allocates I/O storage cache space to each unikernel independently. Fig. 2 shows the overview of resource management in a typical hypervisor. The hypervisor is responsible for an allocation of computing, communication,

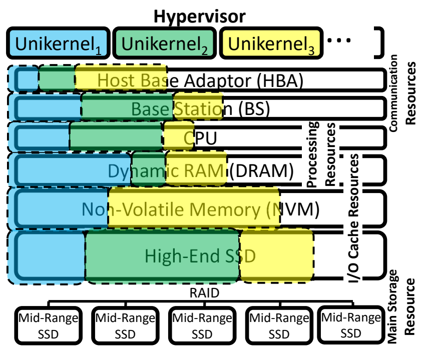


Fig. 2: Resource management of hypervisor in a typical virtualized system.

memory, and storage resources to each unikernel regarding predefined requirements and runtime demands of each VM. The unikernels have different processing and storage demands. Hence, an efficient resource allocation regarding the demands of each unikernel and its priority is crucial. In this paper our major focus is managing I/O cache resources (cache size allocated to each unikernel).

III. Problem Definition

Here, the problem definition is to find the optimum cache size for every unikernel to minimize the objective function. We consider our objective a function of average latency and frequency of accesses in each unikernel. In the objective function, the frequency of accesses is multiplied by the average latency, so the weight of each unikernel in our objective is proportional to its data access rate. This way, we prioritize the latency of unikernels with high data access rate. Accordingly, we formulate the problem as:

$$\min \sum_{i=1}^{N_U} A_{U_i} \times F_{U_i} \tag{1}$$

$$s.t. \quad \sum_{i=1}^{N_U} C_{i,j} \leq S_j, \quad \forall j \in \langle 1, N_l \rangle ,$$

where N_U is the number of the unikernels, A_{U_i} is the average latency of unikernel i, F_{U_i} is the access frequency of the unikernel i, N_l is the number of cache memory layers, $C_{i,j}$ is the cache size allocated to the unikernel i from cache memory layer j, , and S_j is the total size of cache memory layer j. The constraint in (1) assures that total cache allocated from each memory layer is not greater than the size of that layer.

IV. Proposed Storage Architecture and Allocation

In this section, we present the proposed priority-aware block data storage architecture for the edge cloud nodes serving the autonomous vehicles. We first discuss two major shortcomings of contemporary edge architectures when they are employed for autonomous vehicles, and present the motivational results and analysis that inspired the proposed architecture. Afterwards, we elaborate the proposed storage architecture and its management policy. Finally, we demonstrate the process of efficient cache size allocation.

TABLE I: Read/Write latency and price per terrabyte of high-end DRAM [22], NVM (Intel Optane Persistent Memory) [23, 24], SSD [24, 25], and HDD [24, 26].

Latency (ns)	DRAM	NVM (PCM)	SSD (Flash)	HDD
Read Lat. (ns)	7.9	70	25,000	5,000,000
Write Lat. (ns)	7.9	500	200,000	5,000,000
Price ($/TB)	41,152	11,176	3,200	693

A. I/O cache challenges in edge nodes serving autonomous vehicles

1) Performance gap between cache layers: Previous I/O cache architectures for virtualized platforms typically use one or two levels of cache, with combinations such as DRAM-SSD and DRAM-NVM [17–21]. However, reducing the latency gap between cache layers can benefit performance, motivating us to propose a novel I/O cache architecture. Table I shows the latency and cost of high-end DRAM [22], NVM (Intel Optane Persistent Memory [23, 24]), SSD (Intel Optane SSD 900p [24, 25]), and HDD [24, 26] available in the market. In the two-layer cache of DRAM-SSD, there is more than 3,000X difference between the read latency of DRAM and SSD, while in the DRAM-NVM architecture, this difference reduces to 9X. Meantime, in DRAM-NVM cache architecture we observe a large read latency difference of 70,000X between NVM and HDD, while this difference is 200X between SSD and HDD. By architecting an efficient three-layer cache built upon DRAM, NVM, and SSD, the maximum read latency difference between two successive cache layers can be reduced down to 357X (between NVM and SSD). To better demonstrate the benefits of three-layer cache architecture, we compare the performance of DRAM-NVM, DRAM-SSD, and DRAM-NVM-SSD caches for a fixed hardware cost. To reduce the cases, we assume fixed DRAM and HDD budget and consider the three budget cases for NVM and SSD: a) all budget for NVM (DRAM-NVM cache), b) all budget for SSD (DRAM-SSD cache), c) half budget for NVM and half budget for SSD (DRAM-NVM-SSD cache). Fig. 3 shows the average latency for mentioned cache architectures for four mixed random read/write (70/30) workloads with extremely low locality (with Zipf 1.01 distribution), low locality (Zipf 1.05), medium locality (Zipf 1.1), and high locality (Zipf 1.2). As Fig. 3 shows, all four workloads benefit three layer cache architecture, while the workload with high locality even shows a greater performance gain, as the workloads with higher locality have a greater chance of cache hit (having its request responded by the cache memory) and benefit more from an efficient cache architecture, while in workloads with very low locality, most of requests are finally addressed by main storage subsystem and the improvements in the cache memory architecture is of less effect.

2) Starvation of high-priority services: In edge nodes serving autonomous vehicles, another shortcoming of existing I/O cache architectures is the starvation of I/O cache resources in high-priority applications, when traffic of edge node is high. While in conventional edge computing applications, the services have usually similar priorities and the objective of storage architectures is reducing average latency of all services, the inherent difference of autonomous vehicle applications

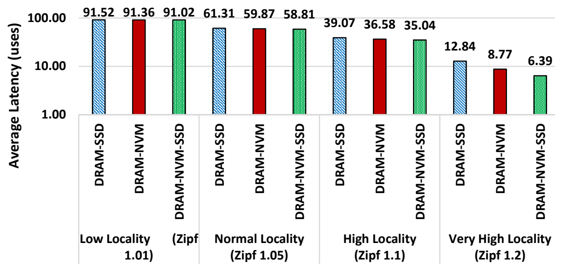
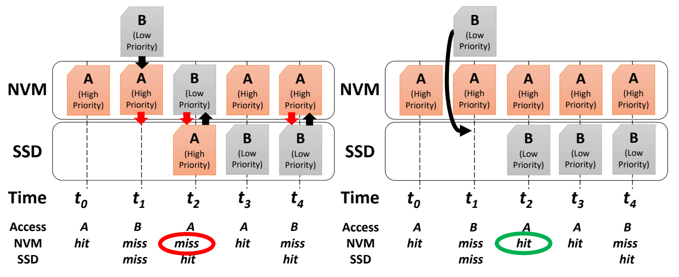


Fig. 3: Comparison of average latency in DRAM-SSD, DRAM-NVM, and DRAM-NVM-SSD cache architectures, considering equivalent cache memory cost.



(a) Conventional cache (b) Priority cache

Fig. 4: Cache memory thrashing problem in conventional edge architectures

in having a combination of time-critical and normal services mandates a caching mechanism aware of service priority. To have a better illustration, Fig. 4 shows a simple example how the conventional cache architectures that do not consider the priority of services encounter cache thrashing[2] and reduce the performance of cache memory. As shown in Fig. 4a, at time t_0, t_2, and t_3, the high-priority block A is requested. However, at time t_1, a request for low-priority block B makes the cache manager to evict block A from NVM to SSD, resulting an NVM cache miss of block A at time t_2. After requesting block A at time t_2, block A is promoted again to NVM and replaces block B, resulting another extra write to NVM and another extra write to SSD for evicting block B from NVM to SSD. By requesting block B again at time t_4, block B is promoted to NVM and block A is evicted again from NVM to SSD. In this case study, the accesses to the cache impose a) an extra NVM cache miss, b) three extra writes to NVM, and c) two extra write operations to SSD, compared to the case we forbid writing low-priority requests into NVM (shown in Fig. 4b.)

In summary, we observed that the performance gap between cache layers results in a huge latency increase in conventional two-level I/O cache architectures, and, concluded that emerging non-volatile memories can fill this gap by providing a latency more near to DRAM when used as a cache layer between SSD and DRAM. We also observed that in existing cache architectures, high-priority services are susceptible to starvation of cache resources in heavy traffics, resulting in reduced hit ratio[3]. An efficient I/O cache architecture for autonomous vehicles should consider the priority of services and make the best use of emerging memories to provide minimum storage

[2]Cache thrashing phenomenon refers to the excessive use of resources or conflicts in the cache memory.

[3]Hit ratio is the number of requests responded by the cache memory, divided by the number of all requests received by the cache memory.

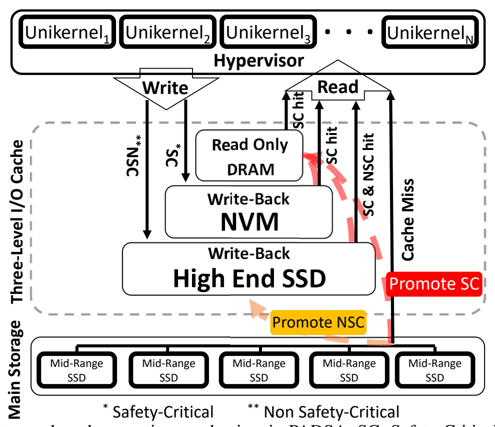


Fig. 5: Write, read, and promotion mechanism in PADSA. SC: Safety-Critical, NSC: Non Safety-Critical.

latency to the time- and safety-critical services. In the next section, we propose our architecture addressing the mentioned issues.

B. Proposed Cache Architecture

The block I/O cache architecture, PADSA, is outlined in Fig. 5. We propose a three-layer cache architecture composed of DRAM, NVM, and high-end SSDs[4]. In the main storage subsystem, either of mid-range SSDs or high-end HDDs are used, while in this work we consider employing mid-range SSDs. As a volatile memory, DRAM cache is configured by read-only[5] policy to forbid data loss in the case of power outages and system failures. In the proposed architecture, DRAM cache is enabled only for safety-critical services (unikernels) and disabled for other services. This way we allocate larger DRAM cache resources to safety critical services and achieve greater hit ratio. NVM cache resides under DRAM level and is configured by write-back policy. Write-back policy buffers both new read and write requests in the cache memory and writes a dirty block back into the storage subsystem, when the block is evicted from the cache. The write-back policy is preferred over write-through, which does not help the performance of write requests and only improves read performance [19][6]. Similar to DRAM cache, NVM cache is dedicated to safety-critical services. Finally, SSD cache is configured by write-back policy and is shared between all services. We also consider *Least Recently Used* (LRU) replacement policy for all three cache layers. In the following, we describe how the proposed architecture manages the incoming read and write requests at the block I/O layer.

[4]SSDs can have diverse performance characteristics, regarding their technology and manufacturing quality, but high-end SSDs in the market can roughly deliver 10x less latency and 10x more *Input/Output Per Second* (IOPS) and bandwidth compared to mid-range SSDs.

[5]In read-only policy, the cache memory is used only for promotion from either lower cache memory hierarchies or main storage, to speed up further read requests. Read-only cache does not keep any write-pending data, as the true data copy is available in the lower cache hierarchies or main storage.

[6]In the write-through policy, write requests are directly sent to either main storage or lower cache hierarchy and the write acknowledgment is sent when writing to the main storage (or lower cache hierarchy) is accomplished. This policy is usually used in volatile cache memories, where the cache memory is prone to data-loss in the case of power outages or system failures.

We also describe the cache eviction and promotion mechanism in the proposed architecture.

1) Write Requests: Upon receiving a new write request of a high-priority service, the block is written on the NVM level, as shown in Fig. 5. In the case of normal services, the block is written on SSD level. This mechanism provides lower write latency to high-priority services. Moreover, the future read requests belonging high-priority services will have more change to be addressed by NVM. Per write operation, cache metadata is updated and the other samples of written data in other cache levels (if any) are invalidated to assure cache coherency, despite the priority of service.

2) Read Requests: Upon receiving a read requests for a high-priority service, the content of cache memory levels is checked from DRAM layer. In the case of cache miss, NVM and SSD layers are respectively checked. In the case of miss, data is finally fetched from the storage subsystem.

3) Promotion: Upon a read request, in the case of high-priority services, the requested data is promoted to the DRAM layer to be accessed by a low latency for the next possible requests. In the case of normal services, only if cache miss happens (in which data is fetched from the storage subsystem), the data is promoted to SSD cache layer. Here, both DRAM and NVM layers are dedicated to high-priority services to increase their hit ratio.

4) Eviction: Eviction logic of each cache layer manages to provide empty cache space for future writes/promotions, when the cache is full. We consider *Least Recently Used* (LRU) replacement policy for all three cache layers, to select the victim cache block and evict it from the cache memory. In the case of DRAM cache, the eviction logic selects the victim block and simply removes its data and metadata. In this case, no write-back to NVM layer is needed, as DRAM layer only keeps *clean* data whose copy exists in either lower cache memory level or storage subsystem. In the case of NVM cache, the eviction logic selects the victim block and checks its *dirty* status. If the block is dirty, it is written back to the SSD layer. Otherwise, if the block is clean, the eviction logic removes the data and metadata of the victim block. Similarly, in the case of SSD cache, the dirty victim blocks are written back to the storage subsystem and the clean blocks are removed.

C. Efficient Cache Size Allocation

Our proposed architecture dynamically manages cache resources allocated to each unikernel during runtime and finds the optimum cache resource allocation regarding the recent workload behaviour. We consider our objective a function of average latency and frequency of accesses in each unikernel. Average latency A_{U_i} of the i-th unikernel in the hierarchical cache is a function of requests filtered in each cache hierarchy determined by cache hit ratio [27], i.e., A_{U_i} is calculated as [27]:

$$A_{U_i} = \begin{cases} L_{r_D} + (1 - H_{D_i}) \times \Bigg(L_{r_N} + \Big((1 - H_{N_i}) \times \\ L_{r_S} + \big((1 - H_{S_i}) \times L_{r_Z} \big) \Big) \Bigg), & U_i \subset U_S, \\ L_{r_S} + (1 - H_{S_i}) \times L_{r_Z}, & U_i \not\subset U_S, \end{cases} \tag{2}$$

where U is the set of unikernels corresponding to the autonomous vehicles served by the edge node, $U_S \subset U$ is the set of safety-critical unikernels, H_{D_i}, H_{N_i}, and H_{S_i} are the hit ratio of DRAM cache, NVM cache, and SSD cache, respectively, allocated to the unikernel i, and L_{r_D}, L_{r_N}, L_{r_S}, and L_{r_Z} are the expected read latency of DRAM cache, NVM cache, SSD cache, and storage subsystem, respectively.

We proceed further to find the optimum cache resource allocation that minimizes the average latency. During runtime, we monitor storage accesses and in specific periods, we estimate efficient cache and reallocate the cache resources (if different from the previous allocation). The average latency is a function of cache hit ratio. The hit ratio of every cache level is calculated by *Miss Ratio Curves* (MRC) [21, 28] that plots the cache miss ratio as a function of cache size by analysing the stack distance[7] when the cache memory uses LRU replacement policy. The MRCs are already available for some known distributions such as *Zipf* distribution. However, as the workload of autonomous vehicles does not resemble the conventional workloads, in the proposed architecture, the MRCs are constructed in specific runtime periods.

V. Performance analysis

In this section, we first present the experimental setup. Afterwards, we demonstrate how the proposed storage cache architecture improves the performance of the edge node in terms of the average latency, the cache hit ratio, and endurance compared to the state-of-the-art.

A. Experimental Setup

Fig. 6 shows the overview of our experiment flow. To evaluate the proposed architecture, we use a cache simulator that post-processes the real block-layer traces of the edge node serving the autonomous vehicles. The experiments are conducted on a Linux Ubuntu server distribution using a Supermicro server with X10DRL-i motherboard, 16 core Intel (R) Xeon (R) E5-2620 CPU, and 128GB RAM. We capture highway tracks data from highD dataset [29]. The number of vehicles connected to the edge node varies over time and the number of vehicles connected as well as the duration of the connection of the vehicle to the edge node is derived from the highD trajectory metadata [29]. We further consider running representative services for each vehicle, while each individual service runs as a single unikernel. The latencies of DRAM, NVM, SSD, and HDD are captured from commercial products, as detailed in Table I.

We use four different representative workloads for the vehicles as summarized in Table II. These workloads include a combination of four basic services S1-S4 of the autonomous vehicles. The basic services are generated using *FIO* benchmarking tool [30]. The characteristics of the storage traces associated to each service is summarized in Table III. S1 represents collision avoidance workload that is considered as a

[7]Stack distance of a memory address, also known as reuse distance, is the number of unique references to other memory addresses before a successive access to the same memory address. In a cache memory with LRU replacement policy, when cache size is $K+1$, all memory addresses with stack distance of K and below have a cache hit.

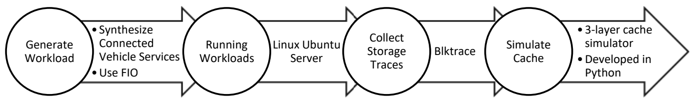


Fig. 6: Experiment flow

high-priority and real-time characteristics. This workload has combination of small read/writes (70%/30%) with relatively high temporal locality (achieved by Zipf 1.2 distribution). S2 represents sensing data upload from the autonomous vehicle to the edge node. This service is of a normal (medium) priority and generates sequential big writes in the edge node. S3 represents map and software update sent from the edge node to the autonomous vehicles. Also, this service is of a normal priority, but generates sequential big read requests in the edge node. Finally, S4 represents any other autonomous vehicle services with normal priority and with random distribution of requests. The storage block accesses for each service are collected using Linux *blktrace* tool [31].

Our proposed architecture is compared to a state-of-the-art storage I/O cache architecture for virtualized platforms, ETICA [19], referred in the results as *baseline*. The ETICA [19] proposes a two-level cache memory of DRAM-SSD for virtualized platforms and proposes considering *Policy-Optimized Reuse Distance* of requests, rather than conventional *stack distance* to calculate the efficient cache size for every virtual machine. This method monitors the workload behaviour and periodically adjusts the cache size of virtual machines during runtime, based on access history. To have a fair comparison, we also consider a three-layer DRAM-NVM-SSD cache in the evaluation of baseline method by extending the ETICA [19] from DRAM-SSD to DRAM-NVM-SSD architecture.

To evaluate our proposed architecture, we simulate the 3-level cache by post-processing the collected block accesses and considering simultaneously running services for vehicles in the sight of edge node. We report three metrics of average latency, hit ratio, and endurance in this section, defined as follows:

Average Latency: The aggregation of average latency achieved by each service multiplied to the access frequency of that service, as shown in 1.

Hit Ratio: The number of requests responded by the cache memory, divided by the number of all requests received by the cache memory.

Endurance: Both SSDs and NVMs fail after a limited number of write operations, while this limit depends on the technology, quality, and size of device. Endurance of a SSD/NVM is proportional to the number of writes committed to SSD/NVM. Here we report the normalized number of block writes (lower value is better) as a representative for SSD/NVM endurance. By considering normalized values, the endurance is comparable regardless of technology and size.

B. Average Latency and Cache Hit Ratio

Fig. 7 shows the average latency of the proposed architecture compared to the state-of-the-art [19] block I/O caching mechanism, shown in figures as a ”baseline”. The results are reported for ”high-priority” services, ”normal-priority” services, and for ”all” services including both normal- and high-priority services. The proposed architecture improves average latency by up to 70%, this improvement is observed for workload_C. For high-priority services, the smallest improvement of 47% is reached for workload_A, while we observe a relatively high improvement of 61% and 70%, compared to the baseline algorithm for workload_B and workload_C, respectively. This observation is justified by the fact that in workload_B, the number of S2 and S3 services is twice higher than in the workload_A. S2 and S3 services, respectively, have a sequential-write and sequential-read nature with normal priority. Intensity of these services in previous works that do not take the priority into account can result in significant performance degradation. We observe even a greater improvement in workload_C due to having the most intensity of S2 and S3 workloads (3x of the workload_A). In the case of workload_D, we observe 53% latency improvement in high-priority services, not as high as improvement observed in workload_B and workload_C. This observations is justified by comparably lower intensity of S2 (sensing data upload) and S3 (map/update download) service types in workload_D. S2 and S3 service types, due to their sequential nature, have a very destructive impact on average latency, as they pollute the cache memory by large data writes and promotions, decreasing the chance of cache hit in high-priority services. While workload_D contains 1X of S4 (other random type services with normal priority), it suffers less from cache pollution compared to workload_B and workload_C, as S4 contains small request sizes with less pollution impact (compared to large request sizes in S2 and S3).

Looking at the normal-priority services, we observe neither performance improvement nor performance degradation in the case of workload_A, workload_B, and workload_C. This observation is justified by the nature of normal-priority services, S2 and S3, appeared in the three mentioned workloads. S2 and S3 respectively have sequential write and sequential read nature with zero locality. Hence, the read requests does not benefit the residence of data in the cache memory, as there is zero chance of re-reference to a block address. Accordingly, all read requests are responded by accessing the main storage subsystem, and the cache memory is of no effect in both proposed and baseline architectures. The only exception is workload_D having a sample of service S4 with random read/write nature and normal priority. This service, as shown in Table III, is characterized by high locality (by using Zipf 1.2 distribution) and can benefit much from cache memory. Hence,

TABLE II: Combination of services types in examined workloads. Each workload is constructed by concurrently running one or more samples of each service type. This table shows the number of samples from each service type appeared in each workload.

Workload	S1: Collision Avoidance	S2: Sensing Data Upload	S3: Map/Update Download	S4: Other
A	1x	1x	1x	-
B	1x	2x	2x	-
C	1x	3x	3x	-
D	1x	1x	1x	1x

TABLE III: Characteristics of autonomous vehicle service types

	S1: Collision Avoidance	S2: Sensing Data Upload	S3: Map/Update Download	S4: Other
Read/Write %	70/30	0/100	100/0	70/30
Access Pattern	Zipf 1.2	Sequential	Sequential	Zipf 1.2
Priority	Real-time	Normal	Normal	Normal

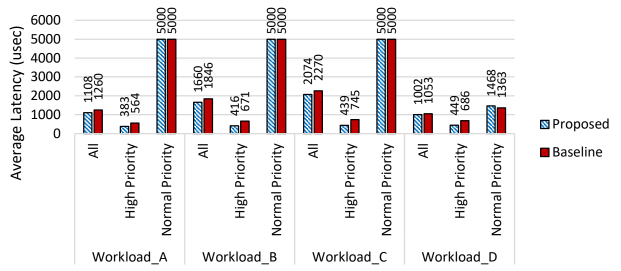


Fig. 7: Average latency for autonomous vehicle services.

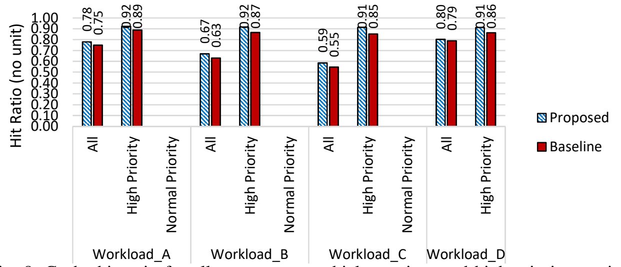


Fig. 8: Cache hit ratio for all autonomous vehicle services and high-priority services.

by dedicating DRAM and NVM cache to high-priority services, we could predict performance degradation in service S4. As shown in Fig. 7, our architecture results in 7.7% performance degradation of normal-priority services in workload_D.

As shown in Fig. 7, the proposed architecture also improves average latency for all services (aggregation of normal- and high-priority services) for all examined workloads. However, the average latency improvement for all services is less than improvement observed in high-priority services. This result is expectable, as prioritizing time- and safety-critical services in the proposed architecture has a negative impact on the average latency of normal services. This negative impact on normal-priority services can affect the overall average latency of all services. In our experiments, however, the impact of performance reduction in normal services is less than improvement achieved for high-priority services, resulting an overall average latency improvement for all workloads. In conclusion, we observe average latency improvement for high-priority services in all examined workloads, while the improvement is greater in the workloads with high intensity of normal-priority services.

Fig. 8 shows the overall cache hit ratio and the hit ratio of high-priority services. As the figure shows, the proposed architecture improves cache hit ratio compared to the baseline by 6%, in average, for the high-priority services. This improvement is translated to, in average, 6% less accesses to the storage subsystem and, consequently, this contributes to latency reduction reported in Fig. 7. By dedicating DRAM and NVM to high-priority services, the proposed architecture also increases the percentage of the read accesses responded in DRAM and NVM cache layers, respectively by 8% and 7%, as shown in Fig. 9. The increase in high-priority accesses responded by two top cache layers, translates to less number of accesses to the SSD and main storage, which both have relatively higher latency compared to DRAM and NVM.

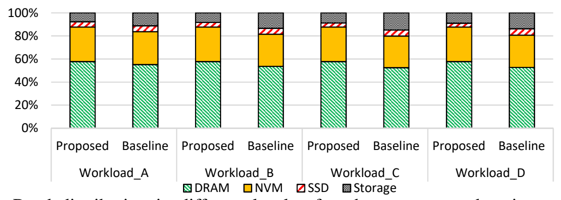


Fig. 9: Read distribution in different levels of cache memory and main storage for high-priority services. This chart shows what percentage of read requests are responded by DRAM, NVM, SSD, and main storage.

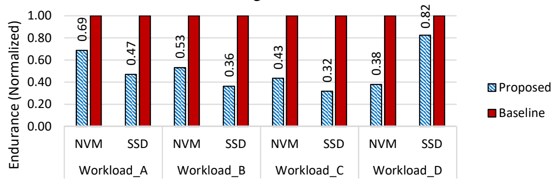


Fig. 10: NVM and SSD endurance for proposed architecture and baseline, normalized to baseline.

C. *Endurance*

Fig 10 shows the endurance of NVM and SSD cache layers, normalized to the baseline, and shows, the proposed architecture improves the endurance by up to 315%. We define the endurance via the number of block writes (lower value is better) and normalize the values by dividing the raw value of the proposed method into the baseline, for each individual experiment. Improving NVM endurance is expected, as the proposed architecture dedicates DRAM and NVM cache to the high-priority services, reducing the number of accesses to both DRAM and NVM. Moreover, less accesses to DRAM reduces the number of evictions from DRAM to NVM, further improving the NVM endurance. While in the proposed architecture, all write requests of normal-priority services are directed to the SSD cache, we still observe endurance improvement even in the SSD layer. This observation is explained by the large number of evictions from NVM to SSD layer in the baseline method. In write-intensive workloads, write requests of normal-priority services result in extra writes into the NVM layer and consequently, extra evictions to SSD.

VI. Conclusion

In this paper, we proposed a novel three-level data storage cache architecture for the edge nodes serving the autonomous vehicles. The proposed architecture prioritizes time- and safety-critical services and allocates the two top-level caches of DRAM and NVM to these services. The architecture further evaluates optimum cache space allocated to each service to minimize the average latency. Our experiment results show that the proposed architecture improves average latency in safety-critical applications by 47% to 70% compared to the state-of-the-art work. Moreover, the cache hit ratio is increased by up to 8% and the endurance of NVM and SSD devices is extended by up to 315% thanks to the proposed efficient cache allocation. In the future, an impact of edge nodes' mobility should be investigated in terms of balancing storage traffic and network latency, achieving the optimum quality of service.

References

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[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] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.

[4] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on scenario-based safety assessment of automated vehicles," *IEEE access*, vol. 8, pp. 87 456–87 477, 2020.

[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.

[6] M. Kishani, M. Tahoori, and H. Asadi, "Dependability analysis of data storage systems in presence of soft errors," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 201–215, 2019.

[7] M. Kishani and H. Asadi, "Modeling impact of human errors on the data unavailability and data loss of storage systems," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 1111–1127, 2018.

[8] M. Kishani, S. Ahmadian, and H. Asadi, "A modeling framework for reliability of erasure codes in ssd arrays," *IEEE Trans. Comput.*, vol. 69, no. 5, pp. 649–665, 2020.

[9] M. Kishani, R. Eftekhari, and H. Asadi, "Evaluating impact of human errors on the availability of data storage systems," in *Design, Automation & Test in Europe (DATE)*. IEEE, 2017, pp. 314–317.

[10] 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.

[11] (2021) Automotive edge computing consortium: General principle and vision. [Online]. Available: https://aecc.org/wp-content/uploads/2020/07/General_Principle_and_Vision_January_31_2020.pdf

[12] J. A. Dias, J. J. Rodrigues, N. Kumar, and K. Saleem, "Cooperation strategies for vehicular delay-tolerant networks," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 88–94, 2015.

[13] 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 *Proceedings of the Workshop on Mobile Edge Communications*, 2017, pp. 7–12.

[14] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, and M. Rossi, "Mobility aware and dynamic migration of mec services for the internet of vehicles," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 570–584, 2021.

[15] J. Plachy, Z. Becvar, E. C. Strinati, and N. di Pietro, "Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users," *IEEE Trans. Netw. Service Manag.*, 2021.

[16] F. Zhou, R. Q. Hu, Z. Li, and Y. Wang, "Mobile edge computing in unmanned aerial vehicle networks," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 140–146, 2020.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] (2021) Column address strobe latency. [Online]. Available: https://en.wikipedia.org/wiki/CAS_latency

[23] (2021) Intel optane persistent memory. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[24] (2021) 3d xpoint: A guide to the future of storage-class memory. [Online]. Available: https://www.tomshardware.com/reviews/3d-xpoint-guide,4747-9.html

[25] (2021) Intel optane ssd 900p. [Online]. Available: https://ark.intel.com

[26] (2021) Seagate enterprise performance 15k hdd. [Online]. Available: https://www.seagate.com

[27] 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.

[28] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal)*, 1970.

[29] 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 *International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[30] (2021) Fio: Flexible I/O tester synthetic benchmark. [Online]. Available: https://github.com/axboe/fio

[31] (2021) blktrace: A block layer I/O tracing tool. [Online]. Available: https://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html