Kubernetes Edge/Cloud Continuum Task Offloading Framework for Vehicular Computing

Alireza Bakhshi Zadi Mahmoodi, Ella Peltonen

Empirical Software Engineering in Software, Systems, and Services, University of Oulu, Finland

Abstract

Cars have significantly been transformed to the point of autonomously driving in complex situations by sensing their surroundings and inferring insights based on sensor inputs. Even though smart cars can process the vast majority of data coming from various in-vehicle-installed sensors such as radars, LiDAR, cameras, and so on, the amount of data processing required is ever-growing, along with the demand for more real-time services for novel driving applications. In addition, sustainability and battery-longevity perspectives appreciate the computation of the vehicle-sensor data to be offloaded to the edge-cloud continuum. This article introduces a Kubernetes-based framework that can be utilized on cloud/edge servers to facilitate various tasks and computation offloading for smart vehicles. The work is ongoing, and we present the preliminary results about the framework's validity by employing an object recognition task on both edge and cloud computing servers to showcase the proposed architecture's feasibility. An incidental finding regarding latency is also presented in the experiment. Moreover, we discuss development challenges related to implementing an edge-cloud continuum on vehicular computing.

Keywords

Vehicular Computing, Task Offloading, Kubernetes, Edge-Cloud Continuum, Microservices, Software-Defined Vehicles (SDVs)

1. Introduction

A lot has changed since the first invention of the three-wheeled gasoline-powered Benz Patent-Motorwagen by Carl Friedrich Benz in 1885. Today's cutting-edge smart vehicles are capable of a multitude of autonomy regarding navigation and manoeuvring, like being able to perceive and understand their surroundings, plan and execute safe roads, handle complex driving situations such as navigating busy intersections, roundabouts, or even merging onto highways seamlessly. Advancements in communication and computing technologies have significantly impacted how cars have morphed into such an autonomous state. Information and Communication Technology (ICT)-enhanced vehicles, which include autonomous vehicles, connected vehicles, and the Internet of Vehicles (IoV), continue to appear increasingly on the horizon [1]. While contemporary smart cars can provide such a degree of autonomy, some considerations should also be given to sustainability and energy efficiency perspectives, especially for battery-operated vehicles. Constant data is provided by myriad sensors, such as LiDAR, radar, cameras, GPS, Inertial Measurement Unit (IMU), microphone, and ultrasonic sensors. The aggregate data can reach as high as 20 TiB [2] and require constant computation to acquire the level of knowledge that is essential for autonomy.

Smart cars are empowered to handle the computational needs for such data via their internal systems, which include high-performance processors like GPUs and CPUs, Field-Programmable Gate Arrays (FPGAs), memory, communication interfaces, etc. However, processing via the internal systems comes at the expense of the limited capacity of the battery that provides the required energy for the entire car, not to mention computation-generated heat that requires cooling, which in turn adds to the overall energy consumption. That is where computation offloading comes into the picture; it can be defined as transferring a task from a resource-constrained device to a more robust system that can handle it officially [3]. By leveraging the transfer, the resource-constrained device can conserve resources to improve its performance and battery longevity [4, 5]. Smart vehicles can also benefit from computation offloading by delegating the required real-time computation to cloud or edge servers, which are more powerful and have "unlimited" capacities, to save battery consumption, spare storage, and access to more computation.

Cloud is one of the candidates that can be employed for computation offloading by smart vehicles. However, some impediments are associated with the cloud, namely latency and network congestion. Edge computing, particularly Multi-access Edge Computing (MEC), is a paradigm considered for vehicular computing environments to reduce latency and improve performance for latency-sensitive applications [6]. MEC servers are typically deployed at the cellular network's edge, giving them low latency and high bandwidth. This

TKTP 2024: Annual Doctoral Symposium of Computer Science, 10.-11.6.2024 Vaasa, Finland

Alireza.BakhshiZadiMahmoodi@oulu.fi (A. B. Z. Mahmoodi); Ella.Peltonen@oulu.fi (E. Peltonen)
0009-0004-3192-7711 (A. B. Z. Mahmoodi); 0000-0002-3374-671X (E. Peltonen)

makes them ideal for offloading tasks from autonomous vehicles, such as path planning, object recognition, and high-definition map processing, which are essential for automated driving [7] when considering the design of Software-Defined Vehicle paradigm as a whole [8]. However, consensus must be reached for scheduling and optimising the offloaded computational tasks from autonomous vehicles to edge/cloud servers.

Motivation: deciding what data to send, how to split tasks, which servers to use, and how to manage everything efficiently are all complex challenges in the task-offloading realm, especially when considering a multitude of underway vehicles in a flowing traffic. Figuring out the best communication methods, infrastructure setup, and technologies like virtualization, containers, microservices, and orchestration all add to the complexity that must be considered and evaluated in simulations and in practice with real-world scenarios. Not much work has been done to tackle the concerns laid out so far, highlighting the research gap for our work.

Research question: how can task-offloading be optimized to improve efficiency and performance, considering the complexities of data transmission, task distribution, server utilization, and the integration of technologies such as virtualization, containers, microservices, and orchestration in real-world scenarios while considering all the concerns mentioned in the motivation section?

Contribution: our final goal is to implement, validate, and benchmark a real-life vehicle-edge-cloud continuum with a real-life vehicle testbed (Figure 1) based on microservices technology orchestrated by Kubernetes (k8s for short; pronounced /keɪts/). This article presents a towards-the-final-goal work-in-progress Kubernetes-based framework that can be employed on cloud and edge servers to facilitate task offloading from smart vehicles. In addition, we discuss the main lessons learned up until now: 1) We showcase the feasibility of the framework that can run through the whole vehicle-edge-cloud continuum, enabling dynamic task offloading. 2) We underline that much of the promised potentiality of the vehicular edge is still not shown in practice with real-world test cases. And 3) we call for considering and utilising such real-world testbeds instead of naive simulations and synthetic data.

2. Literature Review

Advancements in automotive technology, particularly connected and self-driving vehicles, have endowed cars with increased computing, storage, and sensor capabilities [9]. Today's cutting-edge cars can control the steering wheel to change lanes, accelerate and

Figure 1: Driving university testbed's vehicle at Oulu during winter, 2023. [1]

decelerate to adjust the speed, assist when parking the car, etc. They can function with greater efficiency compared to vehicles driven by humans, smoothly accelerating and decelerating while keeping a safe following distance [10]. This heightened efficiency can result in shorter travel duration and decreased fuel usage, alleviating traffic congestion and reducing greenhouse gas emissions [11]. The Internet of Vehicles (IoV) is envisioned as a decentralized transportation network that can autonomously determine how to transport passengers to their intended destinations efficiently [12].

Autonomous self-driving vehicles can retrieve extensive image and map data from the cloud, eliminating the need to store or process this data locally [13]. While both Intelligent Transportation Systems and connected vehicles can benefit from the scalability and cloud's on-demand nature, restrictions in the network infrastructure connecting to cloud servers can hinder certain services, particularly those that demand ultra-low latency or high bandwidth [14]. With many benefits coming from the cloud, we still face some challenges that are crucial for some applications in autonomous vehicles that need to be tackled. These applications could be categorized into either of three interactive, real-time, or auxiliary applications — an example of an auxiliary application is a system diagnostic for error prediction [15].

Since the connection to the cloud is possible through the Internet, latency and network congestion become worrying factors for latency-sensitive applications because most autonomous driving applications are delay-sensitive and are required to return the result in a short period [16]; like Simultaneous Localisation And Mapping (SLAM) which requires the result to be ready within five milliseconds [17]. The conventional vehicle-cloud offloading makes tasks challenging to complete in time [16]. Furthermore, the wide-area network (WAN) delays render it unfeasible to widely

[1]Original image by Mikko Törmänen. © 2023 Mikko Törmänen.

introduce advanced services like augmented reality and virtual reality [18].

Tang et al. [16] provide a container-based offloading module framework that is composed of an offloading decision module, offloading scheduler module (aka node coordinator), and edge offloading middleware (aka offloading service middleware). The offloading decision module resides in the vehicle and is responsible for whether to offload a service to the edge server or not. This component checks for three criteria: Is there enough computing power in the edge server to handle the offloaded application? Is the energy consumption of the offloaded task less than that of the task executed in the vehicle? Is enough memory available at the edge server to handle the offloaded task? The offloading scheduler module manages multiple edge servers within a valid scope via the service management module responsible for monitoring edge servers. Edge offloading middleware resides in the edge servers and is responsible for providing the requested services via launching containers. The authors use a greedy algorithm to maximize the utility of the Multiple Multidimensional Knapsack Problem (MMKP)-modelled problem and show that millisecond-level offloading is possible on the edge. Their unrealistic evaluation is based on simulated and lab-generated data for the suggested MMKP-modeled problem. Moreover, there are a lot of complicated intermediary modules that exist on both the client (vehicle) and server side. In contrast, we try to eliminate the single point of failure in Tang et al.'s architecture by leveraging the k8s cluster as it can manage an "unlimited" number of compute resources under its orchestration control and provide High Availability (HA) in case any control node fails. In other words, instead of having one server to control other servers, multiple servers can be configured to coordinate other servers, hence providing High Availability (HA). In addition, in the proposed architecture, smart cars are ignorant of the framework's intricacies; this means they don't need to communicate back and forth between various edge servers and keep track of their capabilities to offload their tasks. This means that smart cars only offload their tasks to the k8s cluster residing in the edge/cloud, and the cluster serves the offloading requests immediately without the need to find the right and capable worker node to handle the task. This instantaneous serving is possible via the ready-to-serve running microservices in the cluster. This rapid availability of the services through containerization eliminates any overhead imposed by other related works in which a car tries to keep track of various conditions and servers before offloading the tasks, which adds to the overall complexity and intermediary levels.

Blieninger et al. [19] describe a management approach for real-time Kubernetes clusters in the automotive mobile edge cloud. They discuss the challenges — like limited computational capabilities, as well as energy, space, and weight constraints — and requirements of future autonomous driving systems and the role of sensor-equipped MECs in preparing driving tasks. They introduce a management prototype to show the approach's feasibility and provide a timing analysis to investigate the introduced overhead. The focus is on the tool chain's potential to extend and enhance computational capabilities for real-time tasks in vehicles. They also discuss the potential challenges in offloading tasks to the MEC, including time delays and connection failures. The proposed approach aims to enable the complete self-sufficiency of both the vehicle and the MEC, ensuring the safety and predictability of task behaviour. It emphasizes using a decentralized MEC designed as a stationary twin of the car, focusing on task reusability and scheduling. Overall, Blieninger et al.'s work is more about managing different clusters, and no specific information about how offloading tasks are to be done is presented by them. Compared to our proposed framework, we eliminate any need for middle-man-like components such as "prototype gateway" presented in [19] as it checks for some criteria before enabling offloading, which can add to the overall latency and unnecessary complexity. As an additional difference, we can point out that for our proposed framework, in each region that is under the coverage of Radio Access Network (RAN), there exists one cluster, which is composed of multiple powerful compute nodes (servers), that handles any offload requests by the nearby vehicles in that region without the need to communicate with other clusters; simply put, we do not see any point in communicating between clusters since any car goes from one RAN-covered region to the other; meaning a car can directly communicate with clusters without the need of an intermediary. This is unlike the one presented by Blieninger et al., where they have multiple clusters that communicate with each other through the middle-man-like "central status aggregation."

3. Method

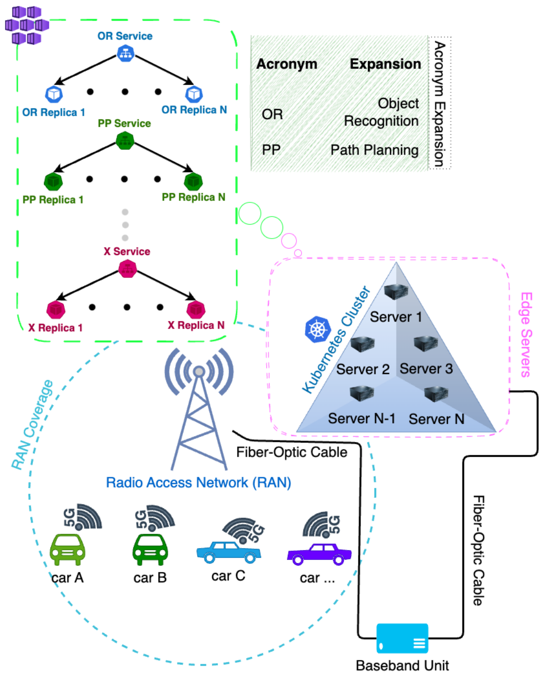
Kubernetes orchestrates containerized applications for scalable, automated deployment and management across the infrastructure. It is considered the operating system of the cloud composed of different components like etcd, API server, scheduler, and control manager, all of which reside on the control plane and are responsible for managing other nodes (worker nodes). The other components, such as kubelet, kube-proxy, and container runtime, are part of worker nodes (compute nodes). Kubernetes, k8s for short, has a modular structure, making it resilient and scalable by allowing dynamic addition or removal of servers as either control plane nodes or worker nodes. The combination of multiple worker nodes and control-plane nodes make up a k8s cluster, which is under the management of control-plane nodes.

3.1. Our Proposed Kubernetes Framework for Computation Offloading

Our presented framework takes advantage of k8s' benefits to provide a structure for computation offloading to cloud and edge servers by smart vehicles. The proposed framework supports the containerization of various applications, scalability, automatic deployment management, and robustness of the back-end services provided by k8s. Figure 2 provides a general view of the offered framework. The back-end servers that form a cluster are all under the orchestration of k8s as either a control plane or a worker node. The control plane is like the brain of the cluster and is responsible for monitoring, managing, and deploying containerized applications on the worker nodes. On the other hand, the worker nodes carry the burden of executing the offloaded tasks requested by smart cars.

As depicted in Figure 2, various services that are needed by smart vehicles, such as object recognition, path planning, blind spot detection, traffic sign recognition, parking assistance, etc., can already be available at a ready-to-serve state in the form of a containerized application running on a worker node to be employed by any vehicle that wants to receive such a service by task offloading. For example, car A wants to receive an object recognition service from the nearby edge server. Car A sends its request to the edge server, managed by k8s in our framework, and receives the requested service already available in one of the worker nodes. At the same time, any other nearby vehicle can receive the same or different services along with other vehicles with their offloaded tasks running in an isolated execution environment on the cluster of the proposed framework.

Multiple advantages can be attributed to the framework empowered by k8s. First, the cars are ignorant of the intricacies involved in how any service is available for computation offloading. Smart vehicles can request any service at any time and provide the required data for the service to be processed by the containerized app running on the k8s cluster in one of the worker nodes. The result of the requested task will be returned to the requester's vehicle after its execution is complete in the cluster's worker node. This ignorance of details is highly important compared to other architectures, such as the one presented in Figure 3 by [16]. There, the car gains access to the nearby edge server through the Node Coordinator, which is a single point of failure and can cause lethal problems, as explained in the following. Whenever the vehicle offloads a task that is



Acronym	Expansion
OR	Object Recognition
PP	Path Planning

Figure 2: Kubernetes framework for computation offloading at the edge. Multiple servers can be orchestrated as a whole entity — known as a cluster — by operating as either the control plane or worker node. Multiple control planes eliminate a single point of failure by becoming highly available, and multiple worker nodes add to the total computation power of the whole cluster. Multiple cars can request their services to be run on any nearby cluster through a wireless connection to the connected Radio Access Network (RAN).

unmet by the already-established edge server, it needs to find a new edge server through the Node Coordinator again. These communications require extra time, which could be detrimental to some applications, such as object recognition, which is necessary for the safety of the passengers inside the vehicle. None of these unnecessary, time-consuming intermediaries is required in our proposed k8s framework presented in this article.

Secondly, the framework is highly scalable and reliable since multiple servers can be added to the cluster as either compute or control plane nodes, eliminating a single point of failure that is present in Figure 3. The third advantage is that the running services in the cluster are instantaneously ready to serve smart cars as soon as the required data for processing is provided over the network. Fourth, the number of running services on the cluster in the ready-to-serve state does not add to the overall computation usage of the worker nodes, which is illustrated in a work by [16] using Docker technologies. Fifth, the offloaded tasks by smart cars

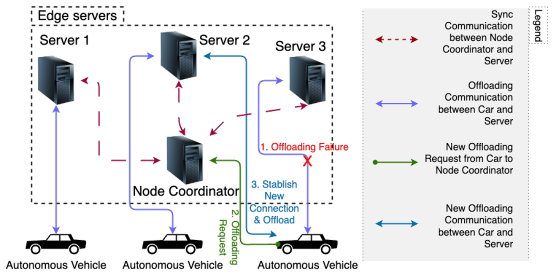

Edge servers
Server 1
Server 2
Server 3
Node Coordinator
1. Offloading Failure
2. Offloading Request
3. Stablish New Connection & Offload
Autonomous Vehicle
Autonomous Vehicle
Autonomous Vehicle
Legend
Sync Communication between Node Coordinator and Server
Offloading Communication between Car and Server
New Offloading Request from Car to Node Coordinator
New Offloading Communication between Car and Server

Figure 3: The classic Node Coordinator architecture relies on a single point of failure, the Node Coordinator, to connect to nearby edge servers. If the established edge server cannot execute the requested task, the vehicle needs to establish a new connection to another edge server via the Node Coordinator. The communication overhead between the vehicle and the Node Coordinator can negatively affect critical tasks like real-time object recognition essential for passenger safety.

run in an isolated environment inside the compute node operating inside the cluster — a feature that comes naturally with a micro-service execution environment. This means that none of the various services running on the same compute node can access each other's resources, such as data, memory, processes, file descriptors, network sockets, etc., which is a positive point from the security perspective.

3.2. The Experimental Setup

For the preliminary evaluation of the feasibility of the framework, two identical environments were set up. The first one in the main building of the University of Oulu's 5GTN test network [2] acted as the edge server. The server is located in the same building with k8s installed to form the required cluster. The cluster provides the necessary environment to run multiple containerized applications, leveraging the server's processing power. Currently, we have one service, YOLOv3 [20], which identifies objects in pictures offloaded via layer-7 HTTP protocol and returns the results as recognized objects in the image. For this particular setup, five identical instances of the set-up service are running simultaneously to handle five concurrent offloading requests. Choosing number five suffices for our small-scale needs. The second environment, the OKD/Kubernetes container cloud called Rahti [3], served as a cloud located 200 km away from Oulu in Kajaani, Finland, and provides the same service as our edge server with same settings and configurations. A client computer, located in Oulu and connected to the University of Oulu's WiFi, acts as a vehicle and sends offloading requests for images to the local edge server and the Rahti cloud. The choice to substitute a computer for a car is because of the lack of some devices for the time being in our testbed. However, the experiment is preliminary, and future works will explore more realistic scenarios.

The preliminary task example: Both environments provide an object recognition containerized service implemented via YOLOv3 [20] algorithm. The example was chosen due to its many possible application areas in autonomous driving and extended service capabilities, as image processing is widely utilised in vehicular computing. The service is employed under the control of the Deployment resource from k8s. Through the Deployment resource, five replicas of the object recognition service are declared to always be available in the cluster for instant availability — meaning at any given time, five object recognition offloading requests can be executed simultaneously in the cluster.

Whenever an object recognition service request is sent along with the to-be-processed data — in this case, images — to the server residing in the edge or cloud, the requested service is provided via one of the already-available-to-be-employed services in the edge/cloud. While the employed service in the edge/cloud server is busy computing the requested task, the other available services can provide the same functionality to new requesters.

In the experiment, the connectivity to the edge server is established through WLAN; again, because of the lack of some devices in the testbed. Smart vehicles should be connected via cellular networks such as 5G, which aims for 1-millisecond latency [21], and future cellular generations with even less latency and higher bandwidth than previous generations. A comparison between different network technologies, as such, is again on the agenda of our future works, as this paper focuses on the feasibility of the proposed framework.

4. Results

The end result for the feasibility of the framework is shown in Figure 4 after requesting offloading task, object recognition in this case, to the framework and receiving the computed result. An offloading request for object recognition takes about 30 seconds, more or less, for an image to be processed by either the edge or cloud servers. This long time taken for object recognition is mainly due to the service being run on a CPU of the worker node instead of a GPU. It has been noted that GPU computation of YOLOv3 can be done within 20 ms [20], which will be experimented with in our future works.

[2]https://5gtn.fi/
[3]https://rahti.csc.fi/

[4]Original image by Mikko Törmänen. © 2023 Mikko Törmänen.

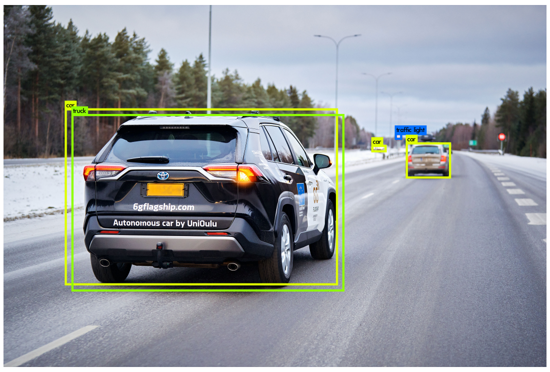


Figure 4: Recognized objects detected by the ready-to-serve object recognition service in the cloud/edge. [4]

Regarding the incidental finding, Figure 5 shows the scatter plot of the latency in each request sent to the edge server (the green plot) and the Rahti cloud (the blue plot), which is less than 500 ms. The x-axis represents time, and the y-axis represents latency in milliseconds. From the plot, we observe that even though the edge server is within the same vicinity, about 100 meters, as the computer requesting the service, most of the time, the latency lies between 45 ms and 75 ms. In contrast, the latency for the cloud lies between 20 ms and 50 ms even though the Rahti cloud is about 200 km away from the requester of the service in the experiment. Standard deviations for both cloud and edge are 23.66 and 14.17, respectively.

Figure 6 shows the latency grouped into various 5 ms intervals for the edge server. It can be observed that most of the time (38 %), the latency lies within intervals of 55 to 60 ms. This insight is counter-intuitive compared to the pie chart of the Rahti cloud shown in Figure 7, with the dominant latency interval (33 %) lying between 25 to 30 ms. The non-obvious higher latency observed in the edge server reveals that the closeness of the requester of service to the edge server can still suffer from latency if the underlying network infrastructure fails to meet the high expectations of the required connectivity.

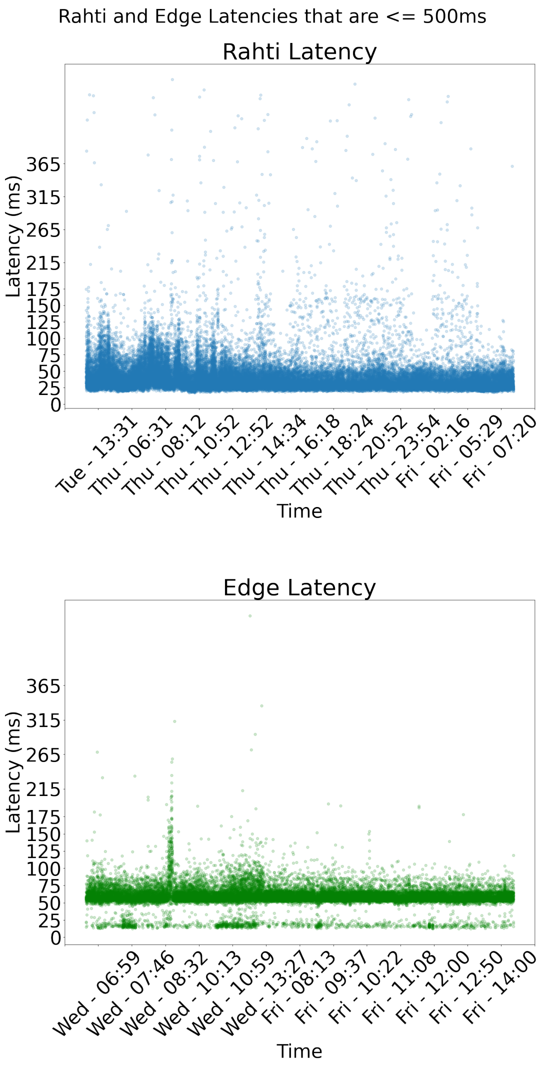


Figure 5: The scatter plot of the latency for both Rahti cloud and the edge server that is less than 500 ms presented by blue and green colours, respectively. Latency for the Rahti cloud is between 20 to 50 ms mostly, while for the edge, it is split between 15 to 20 ms and 45 to 75 ms; it appears as if there is a white line cutting through these two intervals for the edge server latency.

5. Discussion & Conclusions

So, earlier on, we highlighted the research gap as the complex challenges in task-offloading for vehicular computing, such as deciding what data to send, how to split tasks, which servers to use and managing these processes efficiently. We then mentioned the escalated complexity that arises from selecting appropriate communication methods, infrastructure setups, and technologies like virtualization, containers, microservices, and orchestration. Then, we identified the research gap concerning optimizing task-offloading for efficiency enhancement and performance improvement while considering all the complexities.

In this paper, we presented a Kubernetes framework architecture for task offloading by self-driving cars to overcome some challenges. The framework requires the edge/cloud servers to have various replicas of containerized services for multiple applications running in a ready-to-serve state, prepared to be employed

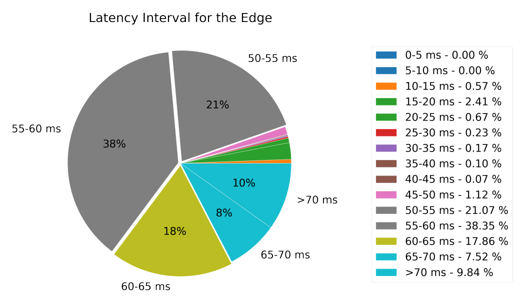


Figure 6: Pie chart of the latency for the edge server. 38 % of the time, latency lies between 55 to 60 ms. 78 % of the time, latency lies between 50 to 65 ms — the three exploded wedges of the pie chart. Counter-intuitively, latency is usually greater than the edge server's counterpart, the Rahti cloud.

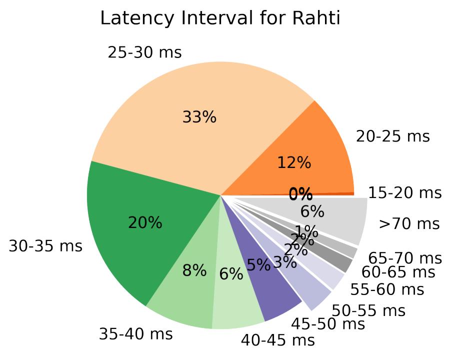


Figure 7: Pie chart of the latency for the Rahti cloud. Most of the time (33 %), latency lies between 25 to 30 ms. 84 % of the time, latency lies between 20 to 50 ms. Non-obviously, most of the time, the latency is less than Rahti cloud's compeer, the edge server.

by smart vehicles. Hence, vehicles offload their computational tasks, such as object recognition, path planning, traffic sign recognition, blind spot detection, etc., to edge/cloud and receive their required results once the data is provided by the smart vehicles to the containerized services.

Based on the literature, numerous advantages can be claimed by using the k8s framework, like scalability, offloading simplicity from the vehicle's side, constant availability of the containerized services, low overhead on the edge/cloud servers for running ready-to-serve containerized services [16], and the isolation of various services while running on the same cluster. In our future works, we will focus on these aspects individually to evaluate whether the Kubernetes-based architecture is right for vehicular task offloading scenarios or if more underlying system design should be studied. Compared to many previous use cases, such as robotics, IoT, and manufacturing, smart vehicle mobility and latency demands are significantly higher. Special considerations should be given to the risk and safety management of the system: a failure over task orchestration cannot, under any circumstances, lead to a fatal failure in the vehicle's operations. In such a case, fatal can become lethal.

Experimental environments for edge and cloud with the same settings were set up as means of validity assessment of the framework. Thus, we can agree that the step towards vehicular edge-cloud continuum architecture is reasonable to consider with similar settings, allowing for more flexibility of dynamic decision-making on the continuum, which was foreseen mainly in visions but less in the practical results [22]. At the same time, our results reveal some non-obvious longer delays with geographically nearby edge servers compared to the cloud. We highlight that, for future work of ours and others, the network specifications and settings should be carefully considered to make a comparison between edge and cloud more reliable. We agree that this is, indeed, easier to control in a simulated environment than in the real-world case. However, we wish to highlight that real-world problems cannot be diminished forever and should be addressed in a timely manner now that we see more and more autonomous cars becoming ubiquitous. In this paper, we can "fix" the network problems as researchers by running optimised test cases for optimal latency. In reality, the wide variety of different network configurations (bad and good ones) of the real world should be considered as a design feature.

Acknowledgments

The work has been supported by the EU HORIZON project CHIPS-JU CIA FEDERATE (grant number 101139749), Business Finland project 6G Visible (grant number 10743/31/2022, and the Finnish Research Council project Northern Utility Vehicle Laboratory Consortium GO!-RI (grant number 352726).

References

[1] C.-M. Huang, M.-S. Chiang, D.-T. Dao, W.-L. Su, S. Xu, H. Zhou, V2v data offloading for cellular network based on the software defined network (sdn) inside mobile edge computing (mec) architecture, IEEE Access 6 (2018) 17741–17755.

[2] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen, A. Y. Ding, A survey on approximate

edge ai for energy efficient autonomous driving services, IEEE Communications Surveys & Tutorials 25 (2023) 2714–2754.

[3] A. Islam, A. Debnath, M. Ghose, S. Chakraborty, A survey on task offloading in multi-access edge computing, Journal of Systems Architecture 118 (2021) 102225.

[4] J. Yang, A. A. Shah, D. Pezaros, A survey of energy optimization approaches for computational task offloading and resource allocation in mec networks, Electronics 12 (2023).

[5] P. K. Nandi, M. R. I. Reaj, S. Sarker, M. A. Razzaque, M. M. or Rashid, P. Roy, Task offloading to edge cloud balancing utility and cost for energy harvesting internet of things, Journal of Network and Computer Applications 221 (2024) 103766.

[6] C. Chen, Y. Zeng, H. Li, Y. Liu, S. Wan, A multihop task offloading decision model in mec-enabled internet of vehicles, IEEE Internet of Things Journal 10 (2022) 3215–3230.

[7] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, X. S. Shen, Toward efficient content delivery for automated driving services: An edge computing solution, IEEE Network 32 (2018) 80–86.

[8] E. Peltonen, A. Sojan, T. Päivärinta, Towards real-time learning for edge-cloud continuum with vehicular computing, in: IEEE World Forum on Internet of Things (WF-IoT), IEEE, 2021.

[9] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, X. Shen, Cooperative task scheduling for computation offloading in vehicular cloud, IEEE Transactions on Vehicular Technology 67 (2018) 11049–11061.

[10] L. Hernandez, M. Hassan, V. P. Shukla, Applications of cloud computing in intelligent vehicles: A survey, Journal of Artificial Intelligence and Machine Learning in Management 7 (2023) 10–24.

[11] C. R. Charles, J. Savier, An overview on hybrid energy storage systems for electric vehicles, International Journal of Electric and Hybrid Vehicles 14 (2022) 56–64.

[12] M. Gerla, E.-K. Lee, G. Pau, U. Lee, Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds, in: 2014 IEEE world forum on internet of things (WF-IoT), IEEE, 2014, pp. 241–246.

[13] K. Goldberg, B. Kehoe, Cloud robotics and automation: A survey of related work, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5 (2013) 13–5.

[14] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder, A. Mouzakitis, A taxonomy and survey of edge cloud computing for intelligent transportation systems and connected vehicles, IEEE Transactions on Intelligent Transportation Systems 23 (2022) 6206–6221. URL: https://doi.org/10.1109/tits.2021.3084396. doi:10.1109/tits.2021.3084396.

[15] Y. Wang, S. Liu, X. Wu, W. Shi, Cavbench: A benchmark suite for connected and autonomous vehicles, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2018, pp. 30–42.

[16] J. Tang, R. Yu, S. Liu, J.-L. Gaudiot, A container based edge offloading framework for autonomous driving, IEEE Access 8 (2020) 33713–33726. URL: https://doi.org/10.1109/access.2020.2973457. doi:10.1109/access.2020.2973457.

[17] J. Van Brummelen, M. O'Brien, D. Gruyer, H. Najjaran, Autonomous vehicle perception: The technology of today and tomorrow, Transportation research part C: emerging technologies 89 (2018) 384–406.

[18] H. Li, G. Shou, Y. Hu, Z. Guo, Mobile edge computing: Progress and challenges, in: 2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud), IEEE, 2016, pp. 83–84.

[19] B. Blieninger, A. Dietz, U. Baumgarten, Mark8s-a management approach for automotive real-time kubernetes containers in the mobile edge cloud, RAGE 2022 (2022) 10.

[20] J. Redmon, A. Farhadi, Yolov3: An incremental improvement (2018).

[21] G. A. Akpakwu, B. J. Silva, G. P. Hancke, A. M. Abu-Mahfouz, A survey on 5g networks for the internet of things: Communication technologies and challenges, IEEE Access 6 (2018) 3619–3647.

[22] A. Y. Ding, E. Peltonen, T. Meuser, A. Aral, C. Becker, S. Dustdar, T. Hiessl, D. Kranzlmüller, M. Liyanage, S. Maghsudi, N. Mohan, J. Ott, J. S. Rellermeyer, S. Schulte, H. Schulzrinne, G. Solmaz, S. Tarkoma, B. Varghese, L. Wolf, Roadmap for edge ai: a dagstuhl perspective, SIGCOMM Comput. Commun. Rev. 52 (2022) 28–33. URL: https://doi.org/10.1145/3523230.3523235. doi:10.1145/3523230.3523235.