Data Management Platform For Smart Orchestration of Decentralized and Heterogeneous Vehicular Edge Networks

BERK AYAZ, University of Antwerp, Faculty of Applied Engineering, Belgium
NINA SLAMNIK-KRIJEŠTORAC, University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Belgium
JOHANN M. MARQUEZ-BARJA, University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Belgium

Intelligent edge orchestration has become a vital component within next generation communication networks, such as 5G. They offer optimal resource allocation and service distribution, hence allow for full utilization of the opportunities provided by those networks. Orchestrators make use of Machine Learning (ML) techniques to determine the most optimal operational decisions, such as deployment and scaling of services, ensuring the quality of the service performance. The training and validation of these models require significant amount of data. However, in such environments we deal with heterogeneous and distributed data sources, in which the data needs to be collected and pre-processed efficiently, and as such, made ready-to-use for these ML models. Hence, in this paper several state-of-the-art data management technologies suitable for edge computing and orchestration are investigated and compared. After investigating the theoretical features of these technologies, they are deployed and tested on the Smart Highway testbed. The strengths and shortcomings of these systems are presented and compared based on the Quality of Service (QoS) requirements for various vehicular services dealing with highly mobile users, i.e., vehicles, which are considered as quite stringent. This hybrid platform, combining several data management technologies, will be used to assist and enrich the research on smart edge orchestration at IDLab and serve as a reference for other interested parties.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Edge Computing, Edge Orchestration, Data Management, V2X, Kafka, Zenoh, ActiveMQ, Pulsar

ACM Reference Format:
Berk Ayaz, Nina Slamnik-Kriještorac, and Johann M. Marquez-Barja. 2022. Data Management Platform For Smart Orchestration of Decentralized and Heterogeneous Vehicular Edge Networks. In *Conference on Information Technology for Social Good (GoodIT'22), September 7–9, 2022, Limassol, Cyprus.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524458.3547254

1 INTRODUCTION

Cloud computing has enabled several benefits to users in various ways. It offers a highly available and performing way of utilizing Information Technology (IT) resources, but its centralized architecture is no longer able to cope with demands of novel real-time services, such as Vehicle-to-Everything (V2X) and Internet of Things (IoT). In order to deal with networking requirements, such as high bandwidth and ultra-low latency, computational workload and data have been offloaded to the edge network [1].

However, this offloading task poses several challenges. For instance, the placement of those edge services is vital in high demanding use cases, e.g., smart factories and autonomous driving, as performance can backfire when the distribution of services is sub optimal. Those applications facilitate a heterogeneous network of devices with a variety of hardware and software specifications [2]. The dynamic behaviour of those networks, with constant topology changes, requires this offloading task to be automated.

Edge orchestration is a significantly challenging multivariate problem, hence recent trends, in both industry and research, opt for the use of Machine Learning (ML) based techniques [7]. ML models require high quality and quantity data originating from networking nodes, e.g., available hardware resources or networking metrics. By training the ML models with the collected data, these models are able to make optimal orchestration decisions, therefore are able to undertake this task. This data, coming from heterogeneous devices, has to be standardized in order to feed them to the ML algorithms, to produce credible and quality decisions for orchestration operations such as service deployment, scaling, termination, and migration. In the operational phase of the model, it is important that the data is delivered as soon as possible, as data that arrives after a certain time threshold might lose its relevancy. Thus, to enable smart orchestration of vehicular edge networks, i.e., to enforce making proactive orchestration decisions that are aligned with the specific service requirements, it is essential to make the orchestrators aware of the service quality, as well as of the consumption of computing and network resources. Given the distributed nature of vehicular edge resources, the designed data management platform needs to be able to cope with resource heterogeneity and strong decentralization.

In order to support the research on ML-based automated and intelligent edge orchestration, in this paper we define the data management platform, which is designed and developed considering the aforementioned challenges. This platform is used to collect data from distributed and heterogeneous edge networking nodes, and feed the orchestrators with the collected data so that they can derive proactive and quality-aware decisions. Initially this data is used for the training of designed edge orchestration algorithms. During the operational phase of the edge orchestrator, i.e., when the ML based orchestrator is able to make optimal decisions, the data management platform is used to collect the current state of the network so the orchestrator is able to make the orchestration decisions. In this phase it is important that the collected data is delivered as soon as possible to make relevant decisions.

To enable making the decisions that are quality and service-aware, we first study the requirements for several use cases in a V2X context, which will be used to verify the feasibility of messaging systems that constitute the data management platform and enable data collection that provides edge orchestrators with the necessary data for making decisions. Second, we tested several messaging technologies and validated them in terms of latency and throughput. Afterwards, we also tested and validated their resource consumption, as the system consists of the resource constrained devices. Based on the strengths and shortcomings of the analysed technologies, we argue for a flexible system which hosts multiple technologies and design our data management system, such that the selection of technologies can be properly made. Some work has been presented in dealing with low latency, high throughput data. For instance, Hugo et al. [8] present an architecture for collecting similar Key Performance Indicators (KPI), combining Kafka with Message Queuing Telemetry Transport (MQTT). However, they obtain results in a simulation environment. In order to validate the performance in a real-word scenario, we deploy our proof-of-concept system on the Smart Highway testbed[1], which is deployed on the E313 highway (Antwerp, Belgium), thereby providing opportunities to perform real-life experimentation with the V2X communication technologies, utilizing Road Side Units (RSUs) as edge computing nodes for distributed and orchestrated V2X service deployments. The RSUs deployed on the E313 highway serve as the edge network of the Smart Highway testbed, thereby hosting several services for various users, such as vehicles and other ML models. The work presented in this paper will further guide us when designing an orchestration architecture, ensuring a robust working data management platform for collecting heterogeneous and decentralized data from vehicular edges.

[1]Smart Highway testbed https://www.fed4fire.eu/testbeds/smart-highway/

Table 1. Survey Comparison

	Apache kafka	Eclipse Zenoh	Apache ActiveMQ	Apache Pulsar	KubeMQ
Message Replay	Yes	Yes	No	Yes	Yes
Throughput	600 MB/s	1.25 GB/s	38 MB/s	305 MB/s	720 MB/s
Latency	5 ms	35 µs	1 ms	25 ms	N/A
Packet Ordering	Yes	Yes	Yes	Yes	N/A
Packet Loss	Low	Low	High	Low	N/A
Resource Footprint	1.2 GB	300 B	200 MB	870 MB	30 MB

2 MESSAGING TECHNOLOGIES

In this section several messaging technologies are briefly explained. These technologies allow the transfer of data from the producer, i.e., a networking node, to the consumer, i.e., an edge orchestrator. These have been selected following a literature study. An overview of specifications and results from other comparative studies [20] [15] [5] [11] can be found in Table 1.

Apache Kafka. Apache Kafka is an open source distributed messaging system developed at LinkedIn, which is designed to process high volumes of event streams [3]. Over the years, it has matured into a robust and reliable data streaming platform and is heavily adopted by key industry players, such as Cisco and Netflix [16]. It is used in various use cases ranging from event monitoring to collecting system metrics. The Kafka ecosystem consists of producers, consumers, brokers and Apache Zookeeper[2]. Records are produced on topics by producers and consumed by consumers. Depending on the configuration, Zookeeper can distribute and/or replicate these topics over multiple brokers, which may be physically separated. The Zookeeper also orchestrates incoming read/write requests from producers and consumers. Kafka implements an append only log file to allow sequential disk I/O operations, which reduces disk seek time [10]. Jacobs [9] even presents results where sequential disk access is faster than RAM read/write. Additionally, Kafka applies use of the zero-copy principal for Unix systems, which reduces the number of copies and system calls. The aforementioned techniques allow persistence on disk space while maintaining high throughput and low latency. Each record within a topic is identified by an offset, which allows consumers to re-read the same log. The retention time of these records can be configured to the needs of the application. Kafka is written in Scala and Java, and offers clients in many popular programming languages.

Eclipse Zenoh. Zenoh was constructed to address data in movement, data at rest and computation in a scalable, efficient and location transparent way. It is designed for edge networks, in particular to deal with its heterogeneity [6]. It provides a pub/sub system with geo-distributed storages, queries and computations while maintaining low latency and high throughput. Zenoh has minimized its network overhead, offering a wired overhead as low as 5 bytes for a data message. Its footprint on the Arduino Uno is 60 KB [17], making it suitable for extremely constrained transport mechanisms, e.g., Bluetooth Low Energy (BLE). Additionally, it offers time and space efficiency while keeping in mind asymmetric systems and interaction models, e.g., extremely constrained devices in sleep mode. Zenoh provides several mechanisms: (i) efficient publish/subscribe with dynamic discovery, wire-level batching and various levels of reliability; (ii) distributing storage over various machines to store and retrieve data automatically; (iii) querying and aggregating tasks with well-defined semantics. Zenoh uses key/value spaces, e.g., (/network1/device1/ram, 11.3), to represent data

[2] Apache Zookeeper is a service which coordinates the configuration and synchronization of distributed services.

within the network. In order to retrieve a piece of data, it is only required to have the key, also called the path, without needing further information about the underlying network. The retrieval is fully carried out by Zenoh itself. Another feature of Zenoh allows the user to register a specific computation, e.g., calculate device CPU usage, to an endpoint, which can be executed by performing a get operation on a selector matching the path, similar to remote procedure calls. The aforementioned abstractions allow developers not to be bothered by retrieval of data and hence can focus on which data to process and how to process them.

Apache ActiveMQ. As of writing this paper, Apache ActiveMQ is one of the most popular messaging brokers, open sourced and maintained by the Apache foundation. It is written in Java and has full a Java Messaging Service (JMS) implementation and supports various messaging protocols such as MQTT. It can be scaled vertically, and brokers can collaborate with each other. This network of brokers can form different topologies, e.g., publish subscribe. ActiveMQ ensures that a message on the broker is consumed exactly once. This is convenient when a consumer is changing its host device due to relocation of edge network functions, as an outcome of orchestration operations. We distinguish the classic broker from the new broker version called Artemis, which becomes the successor in newer versions of ActiveMQ. It also provides client APIs in different programming languages to create publish and subscribe applications.

Apache Pulsar. Apache Pulsar is a cloud-native platform for distributed messaging and streaming. Its core features include: (i) low latency with durability; (ii) geo-replicated persisted messages with IO-level isolation for read-write operations; (iii) high horizontal scalability across nodes [4]. Although [20] presents results where Kafka performs better, a more recent study shows results where Pulsar comes on top [14].

KubeMQ. KubeMQ is defined as a Kubernetes Message Queue Broker [12]. It is designed for dynamic microservice environments, such as fog networks, offering a variety of messaging patterns. They focus on messaging systems deployed in Kubernetes environment. It has little information available, however KubeMQ [11] claims to have better performance compared to most messaging technologies in a Kubernetes environment. In contrast to the previously described technologies, KubeMQ is not open source and is paid for commercial use.

3 USE CASES

In this section we present several use cases from our literature study that is used to define our KPIs. These use cases are centred in a V2X context, because vehicles are highly mobile users that make orchestration of edge services and resources more challenging. Within these use cases, the data management system is used to collect data from the environment, e.g., CAN-data from vehicles or system metrics from RSUs. This data is used to improve the important decision-making processes that affect service quality. The defined KPIs are used to study and validate the suitability of different message queuing technologies for building a robust data management platform for intelligent orchestration. An overview of the latency and bandwidth requirements are found in Table 2.

Use Case 1: Cooperative adaptive cruise control (CAAC) based platooning. Marquez-Barja et al. [13] presents the requirements for the implementation of Cooperative adaptive cruise control (CAAC) based platooning. In this use case, the leading truck of a truck platoon is teleoperated while the following trucks are automated to follow. This application has induced notable interest for the trucking business as it can alleviate the lack of truck drivers. This use case combines several fields, i.e., adaptive cruise control, automated lane-keeping, and CV2X communication, hence this application consists of several communication parts, each with its own requirements. Only the Vehicle Control Interface and the uplink HD Camera Streaming are considered as these are the most important and demanding parts

Table 2. Use Case Requirements

Use Case	Bandwidth	Latency
Use Case 1 - Vehicle Control Interface	2Mbps	<35ms
Use Case 1 - HD Video Stream	5-25Mbps	<50ms
Use Case 2 - Emergency Brake Warning	2-4Mbps	<120ms
Use Case 3 - Traffic Jam Warning	2-4Mbps	<2000ms

of the system. The requirements in Table 2 show that the latency for the Vehicle Control Interface and HD Camera Streaming should not exceed 35 and 50 ms respectively, otherwise the calculated commands may arrive too late and can lead to catastrophic consequences, e.g. vehicle collisions.

Use Case 2: Emergency Brake Warning. Thomas et al. [19] presents use case requirements where vehicle A that is following vehicle B will be alerted when vehicle B is breaking heavily. When vehicle A is alerted in time, the driver or the vehicle can react in time to avoid collision with vehicle B. The data management platform will be used to communicate data, e.g., when breaking, between vehicles and other networking nodes. In order to avoid catastrophic consequences, the maximum allowed latency is 120 ms, as shown in Table 2.

Use Case 3: Traffic Jam Warning. In this use case, a vehicle is warned about an approaching traffic jam. Although, commercial deployments to display traffic jams on the map are present, e.g., Google Maps and Waze, real time warnings when approaching a traffic jam have not yet been provided. Thomas et al. [19] presents the requirements for urban, rural and highway scenarios. We select the urban scenario as this is the most demanding one, due to the high dynamicity of traffic, hence a high number of network topology changes.

4 EXPERIMENT AND RESULTS

In this section, we evaluate the performance of messaging technologies introduced in Section 2, i.e., Apache Kafka, Eclipse Zenoh, Apache Pulsar and Apache ActiveMQ. These technologies are used to transfer data between networking nodes within the system. Kafka is evaluated on both the Java implementation, using the Spring Framework, and a Python implementation, with the Confluent kafka Python library. This way we can compare performance differences for different platforms. However, they will still be using the same broker. KubeMQ is not evaluated because of its proprietary code base. The goal of this evaluation is to verify the feasibility of messaging systems with respect to the requirements presented in section III. Accordingly, the time to produce and consume a variable number of packets, the end-to-end latency, CPU usage, RAM usage and power consumption is measured.

4.1 Test setup

In order to obtain realistic results for V2X use cases, the Proof-of-Concept (PoC) has been deployed on The Smart Highway testbed. This testbed is designed for V2X research and hosts two types of networking nodes, i.e., On Board Units (OBU) and RSUs. The RSU is installed on a fixed location along the E313 highway in Antwerp and communicates with a mobile OBU, which is equipped with a 4G mobile transceiver. The hardware specifications for the RSU and the OBU can be found in Table 3. The applications are deployed on LXC Containers running Ubuntu 20.04.4 LTS. For each technology, the PoC setup consists of a producer, a consumer and if necessary, a broker. In this PoC, the producer

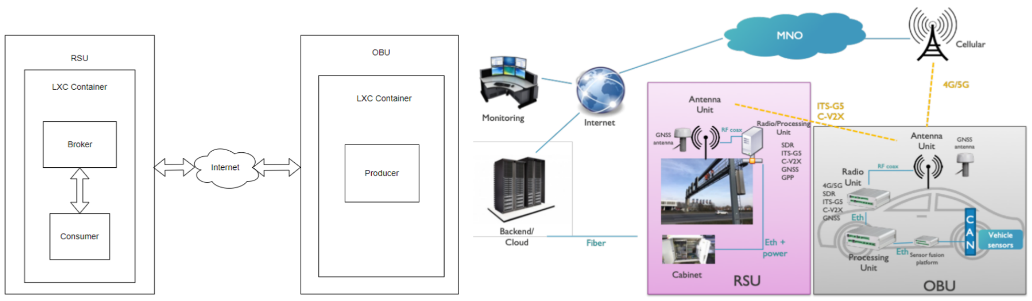


Fig. 1. Smart Highway Test Setup.

Table 3. Hardware Specification

	CPU	RAM
OBU	Intel Xeon E5-2620 @ 2.1 GHz	32 GB RAM
RSU	Intel(R) Core(TM) i7-8650U	8 GB RAM
Desktop	Intel(R) Core(TM) i5-8600K	16 GB RAM

application is deployed on the OBU, whereas the consumer is deployed on the RSU. This can however be deployed on any node to collect data from. If the technology requires a broker, this is also be deployed on the same RSU.

4.2 Defining KPIs

The first considered KPI is T_{comp}, which is the time available for the edge orchestration operation, e.g., calculated on which node to deploy a service. This is calculated as follows:

$$T_{comp} = T_{tot} - T_{com} \tag{1}$$

Here, T_{tot}, the total allowed latency for a use case, is obtained from Table 2 and T_{com}, the communication latency, which is influenced by the latency introduced by the investigated messaging technology, is presented in Section IV. In general, it is desired to keep the communication latency as low as possible in order to have more time for additional computations within the use case, e.g., vehicle collision detection calculations. Throughput is one of the key requirements that has to be satisfied to be able to deploy the considered vehicular use cases. Hence, the second considered KPI is throughput, which is defined as follows:

$$Throughput = \#packets/second \tag{2}$$

Next, the resource consumption, i.e., power, CPU and RAM, is also an indicator for our system. This is because the network may consist of resource constrained devices, hence it is important to have an insight into their computational load in order to balance the service deployments and avoid overloading particular nodes.

4.3 Results

In order to study the throughput for each messaging technology, the time to produce and consume a number of packets has been measured. For each technology, the producer, which is deployed at the OBU, sends a number of packets in succession to the consumer, which is deployed at the RSU. By conducting this test with different number of packets, we can also examine its scalability. The payload size is 256 bytes, which is selected based on the requirements presented in [19]. We have used the default configuration for all technologies. The results are shown in Figure 3. Due to its high values, the results for Pulsar are presented in Figure 4. It is noticed that Kafka performs the best when handling high amounts of packets. Although the performance of Zenoh is similar for lower number of packets, it is not able to scale as well as Kafka. Results also show that Kafka implementations using different programming languages yield similar results.

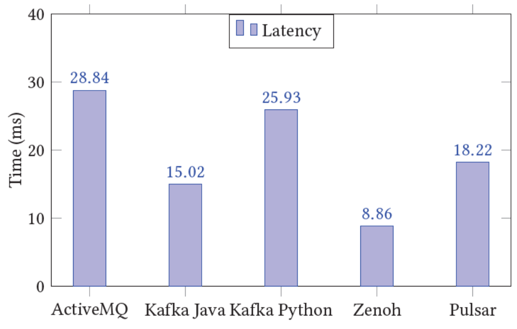
In order to measure the end-to-end latency, a packet is sent from producer to consumer and back again to the producer. The packet size is again 256 bytes, to reproduce the use cases mentioned in previous section. We take the average of 50 packets to minimize the effect of outliers. The results are presented in Figure 2a. The results show that Zenoh has the lowest latency averaging almost 8 ms, followed by Kafka, Pulsar and ActiveMQ respectively. It can also be seen that the Java implementation of Kafka has significantly lower latency values as opposed to the Python implementation. This is probably due to the fact that the Java implementation outperforms the Confluent Kafka Python library, which is a wrapper around the Kafka C implementation, i.e., librdkafka.

In order to measure the RAM usage, CPU usage and power consumption, a test application is deployed on the system. In this application, every second the producer collects system information, i.e., total CPU usage and RAM usage of all processes combined and produce it as a JSON string on a topic which is then consumed by the consumer.

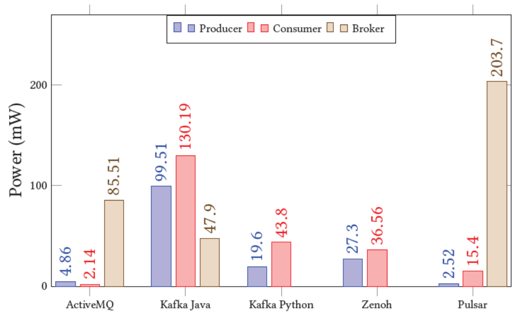
A Python application using the powertop library is deployed and used to measure the power consumption for each technology. The powertop library makes use of the UNIX command line tool called Powertop, which uses the Intel Running Average Power Limit (RAPL) feature to improve measurement accuracy. Due to problems with permissions on the containers hosted on the Smart Highway testbed, the power consumption tests have been conducted on a different system, whose specifications can be found in Table 3. While the application performs its actions, we measure the consumed power for 10 seconds for each individual actor. Results show that Kafka Java utilizes significantly more power compared to the other technologies. Furthermore, results show that ActiveMQ clients use the least power out of the tested technologies. The drawback of ActiveMQ is that the broker uses considerably more power, even more than the Kafka broker, which causes the whole application to be more power consuming than Zenoh and the Python implementation of Kafka. Similarly, the clients of Pulsar are power efficient, however its broker uses a huge amount of power, which results in a higher overall power consumption than ActiveMQ.

Using the same test application, the average percentual CPU usage over 10 seconds has been measured. The results, shown in Figure 2c, are similar to the power consumption, where Kafka Java uses the most processing resources. Due to lack of precision of the tool, results show 0.0 % CPU usage for the ActiveMQ clients, however it still uses some resources. Again, we see that the client consumption is very low for ActiveMQ and Pulsar, however due to the high demanding broker the overall consumption exceeds Zenoh and Kafka Python. Java Python again uses less CPU than the Java implementation, yielding similar values to Zenoh.

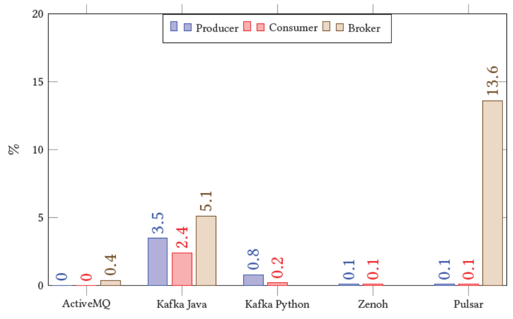
Results presented in Figure 2d show that Kafka Java uses the most memory, again similar to the previous resource usages. The clients for ActiveMQ, Kafka python and Pulsar are on par in terms of RAM usage, and although the RAM usage for Zenoh is higher than the prior technologies, the brokers for these technologies yields a higher overall RAM usage.



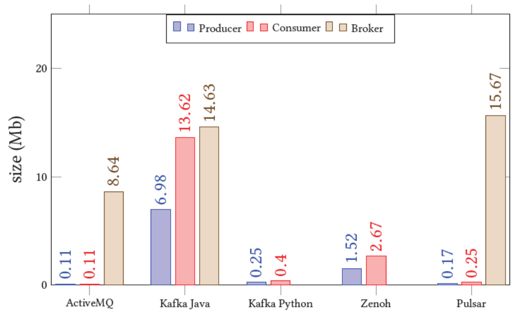
(a) End-To-End Latency in milliseconds.



(b) Power Consumption in milliwatt.



(c) Average CPU Usage over 10 seconds.



(d) RAM Usage in Megabits.

Fig. 2. Results of benchmarking different data collection systems.

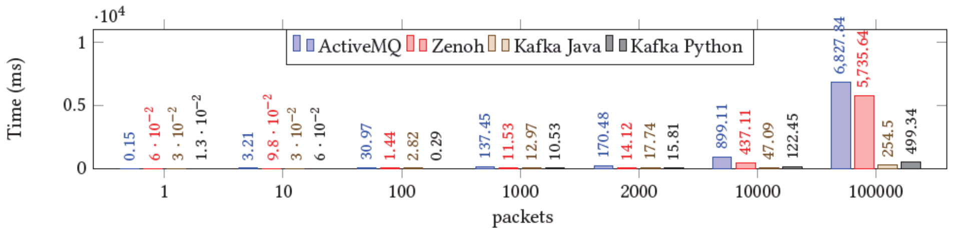


Fig. 3. Time to send N packets in milliseconds.

5 DISCUSSION

5.1 Linking back to the use cases

Here we focus on the use case requirements studied in Section 3, and discuss how the data management technologies we tested and evaluated in Section 4 can be used for creating robust data management systems for decentralized and highly heterogeneous vehicular edge environments.

In particular, the vehicle control interface requires a bandwidth of 2 Mbps, which is a total of 977 packets per second with a payload of 256 bytes, while not exceeding a latency of 35 ms. The test results in Figure 3 show that Kafka Java, Kafka Python and Zenoh are capable of producing and consuming 1000 packets in 12.97, 12.89 and 11.53

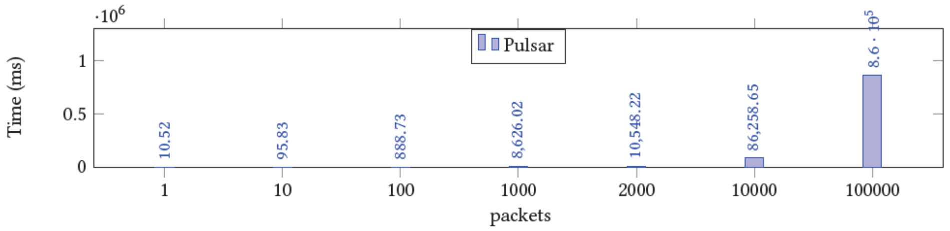


Fig. 4. Time to send N packets in milliseconds.

ms respectively. Following Equation 1, the system yields a budget of respectively 22.03, 22.11 and 23.47 ms, for the computation of commands for the teleoperated trucks. We cannot conclude whether this budget is sufficient to perform these computations and while this is beyond the scope of this paper, this can be investigated further.

The HD camera streaming requires a bandwidth between 5 and 25 Mbps, which corresponds to 2442-12208 packets, while not exceeding a latency of 50 ms. The test results in Figure 3 show that the technologies are not capable to satisfy the requirements for this service. The results show that Kafka Java performs best by producing and consuming 10000 packets in 47.09 ms, which means that with the additional 2208 packets sent, it will exceed the maximum allowed latency of 50 ms, thereby leaving no time budget for additional decision-making computations performed by the orchestrator. A solution might be to reduce the quality of the video or to use more efficient compression algorithms to reduce size and eventually the throughput required.

For the emergency brake warning use case, a throughput of 4 Mbps is required with a maximum latency of 120 ms, which corresponds to 1954 packets of 256 bytes payload. From Figure 3 it is seen that Kafka Java, Kafka Python and Zenoh are suitable for this use case. The remaining computation budget, following Equation 1, yields 102.26, 104.28 and 105.88 ms for Kafka Java, Kafka Python and Zenoh respectively. Again, it cannot be concluded whether the remaining computation budget is sufficient.

For the traffic jam information, a throughput of 4 Mbps is required with a maximum latency of 2000 ms, which corresponds to 1954 packets of 256 bytes payload. In this use case, with the presence of sufficient amount of resources, apart from Pulsar, all the tested technologies satisfy the requirements. If the resource consumption is also a requirement, we advise the use of ActiveMQ. However, we also suggest placing the broker on the edge computing node that has the highest amount of available resources.

In general, if the consumers are deployed in the cloud, where resource consumption is not an issue, we advise the use of Kafka when designing and developing robust data management systems. Besides its performance, Kafka offers various useful features. All the considered use cases are situated in an V2X context, hence, it is assumed that the data sources are heterogeneous, i.e., CAN-data from different car manufacturers, which means the data is not standardized. For this type of situations, one can opt for the use of the Kafka Streams API, which offers an efficient way to aggregate data from different sources into a standardized format. However, it is important to keep in mind that when placing the consumers and brokers in the cloud, the performance in terms of latency may deteriorate. Furthermore, as of Kafka 2.8.0, it has the opportunity to replace the Zookeeper with an internal quorum service [18]. This allows administrators to achieve an easier configuration, as they no longer have to learn, configure and maintain Zookeeper. Although the current version of Zookeeperless Kafka is in early access and is not suited for the production environments, the authors of this solution expect a complete switch by the end of this year. This removal will most likely reduce the resource footprint of the broker. Additionally, its distribution and scalability features will be improved.

In case the nodes are deployed on the edge, where they have less available resources, the user can use the Python implementation of Kafka, as results show that it still performs well while maintaining a fairly low resource footprint. Note that several features, such as Kafka Streams, are currently not available for Python implementations. Although ActiveMQ clients resource consumption is suitable for overly constrained devices, its performance, in Figure 3 and Figure 2a show that it is not able to handle the requirements of most of the aforementioned use cases.

5.2 A Hybrid Architecture

Regardless of the superior performance of Kafka presented in the theoretical study followed by a realistic implementation on the testbed, the vehicular edge are usually resource constrained and require data management technologies that are more lightweight. Besides the technology, the configuration of the technology plays a significant role in the performance, e.g., pre-fetch size configuration in ActiveMQ. This means that an edge node should be able to change its messaging technology and/or configuration according to the orchestrator commands. These commands are calculated based on system metrics, networking metrics and the use case requirements.

In order to implement this feature, we propose the following guidelines. We propose designing and creating an application that is able to switch between messaging technologies based on incoming Representational State Transfer (REST) calls. For this, the orchestrator sends a JSON with an HTTP POST request to the service. This JSON contains all the required parameters in order to make this switch. A drawback of this approach is that the consumer and producer applications require to have all the libraries for all technologies, which will require more disk space. Initially, this is implemented to support only Python clients, but can be extended to run non-Python clients as well. The detailed design and performance of such a hybrid system for data management combining various messaging technologies is part of our future work.

6 CONCLUSION

In this paper, different messaging technologies have been researched and based on the conducted tests, we provide guidelines on designing an architecture for robust data management systems that facilitate the edge network orchestration. It started by giving an overview of the analysed messaging technologies and presented a set of KPIs based on use cases in the V2X context. The technologies have been deployed and tested on the Smart Highway in order to demonstrate its performance and analyse its feasibility for creating a data management system that will support orchestration of the given use cases.

The results show that Pulsar is not feasible for any the presented use cases. Furthermore, the results show that ActiveMQ is only feasible for the traffic jam information use case. All other tested technologies satisfied the requirements for vehicle control interface, emergency breaking and traffic jam information use cases. There was no technology that met the requirements of the HD Camera Streaming use case. The tests have been conducted with default configuration, hence when this configuration is optimized for a particular use case, results may improve. Furthermore, we tried to keep the test scenarios for different technologies to be as similar as possible, which means test scenarios may be favourable for particular technologies. For example, Zenoh is designed specifically to deal with mobile nodes, whereas the test setup consisted of stationary nodes. From the selected use cases and the conducted tests, the total allowed latency and the total communication latency is known. In order to confirm the feasibility, future work needs to be done on the total computation time required for edge orchestration. In order to check feasibility of use cases where only the uplink latency requirement is given, one way latency metrics should be investigated. Following our tests and the use cases, we conclude that the proposed platform requires a mechanism to change the used messaging system and/or its configuration. Benchmarking these messaging technologies with optimized configurations, as well as the design and creation of a hybrid data management system that dynamically selects the optimal messaging technology is part of our future work.

ACKNOWLEDGMENTS

This work has been performed in the framework of the EU H2020 project VITAL-5G (grant agreement No. 101016567), and the H2020 5G-Blueprint project (No. 952189), and it has been also supported by the Flemish Ministry of Mobility and Public Works (MOW) and the regional Agency for Roads and Traffic (AWV), Belgium.

REFERENCES

[1] Ahmad Alhilal, Tristan Braud, and Pan Hui. 2020. Distributed Vehicular Computing at the Dawn of 5G: a Survey. *CoRR* abs/2001.07077 (2020). arXiv:2001.07077 https://arxiv.org/abs/2001.07077

[2] 5G Americas. 2021 [Online]. 5G Edge Automation and Intelligence. https://www.5gamericas.org/wp-content/uploads/2021/10/5G-Edge-Automation-Optimization-InDesign-1.pdf

[3] Apache. 2022. *Apache Kafka.* https://kafka.apache.org/

[4] Apache. 2022. *Apache Pulsar.* https://pulsar.apache.org/

[5] Eclipse. 2021. *Zenoh performance: a stroll in Rust async wonderland.* https://zenoh.io/blog/2021-07-13-zenoh-performance-async/

[6] Eclipse. 2022. *Eclipse Zenoh.* https://zenoh.io/

[7] Tom Goethals, Bruno Volckaert, and Filip De Turck. 2021. Enabling and Leveraging AI in the Intelligent Edge: A Review of Current Trends and Future Directions. *IEEE Open Journal of the Communications Society* 2 (2021), 2311–2341. https://doi.org/10.1109/OJCOMS.2021.3116437 doi: 10.1109/OJCOMS.2021.3116437.

[8] Åsmund Hugo, Brice Morin, and Karl Svantorp. 2020. Bridging MQTT and Kafka to support C-ITS: a feasibility study. In *2020 21st IEEE International Conference on Mobile Data Management (MDM).* 371–376. https://doi.org/10.1109/MDM48529.2020.00080 doi: 10.1109/MDM48529.2020.00080.

[9] Adam Jacobs. 2009. The Pathologies of Big Data. 1–12.

[10] Jay Kreps. 2011. Kafka : a Distributed Messaging System for Log Processing.

[11] KubeMQ. 2022. *Compare KubeMQ.* https://kubemq.io/compare-kubemq/

[12] KubeMQ. 2022. *KubeMQ.* https://kubemq.io/

[13] Johann M. Marquez-Barja, Dries Naudts, Vasilis Maglogiannis, Seilendria A. Hadiwardoyo, Ingrid Moerman, Matthijs Klepper, Geerd Kakes, Xiangyu Lian, Wim Vandenberghe, Rakshith Kusumakar, and Joost Vandenbossche. 2022. Designing a 5G architecture to overcome the challenges of the teleoperated transport and logistics. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC).* 264–267. https://doi.org/10.1109/CCNC49033.2022.9700565 doi: 10.1109/CCNC49033.2022.9700565.

[14] Merli, Matteo and Li, Penghui. 2022. *Apache Pulsar vs Apache Kafka 2022 Benchmark.* Technical Report. Stream Native.

[15] Sanika Raje. 2019. *Performance Comparison of Message Queue Methods.* dissertation. The University of Nevada. https://digitalscholarship.unlv.edu/thesesdissertations/3746/

[16] Jun Rao and Jay Kreps. 2016. *Powered By.* https://cwiki.apache.org/confluence/display/kafka/powered+by

[17] Nina Slamnik-Kriještorac, Paola Soto-Arenas, Miguel Camelo Botero, Luca Cominardi, Steven Latré, and Johann M. Marquez-Barja. 2022. Realistic Experimentation Environments for Intelligent and Distributed Management and Orchestration (MANO) in 5G and beyond. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC).* 943–944. https://doi.org/10.1109/CCNC49033.2022.9700659 doi: 10.1109/CCNC49033.2022.9700659.

[18] Ben Stopford and Ismael Juma. 2022. *Apache Kafka Made Simple: A First Glimpse of a Kafka Without ZooKeeper.* https://www.confluent.io/blog/kafka-without-zookeeper-a-sneak-peek/

[19] Linget; Thomas. 2021. *Use Case Implementation Description (UCID).* Technical Report. 5G Automotive Association.

[20] Chandar Vinoth and Nikhil Alok. 2020. *Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?* https://www.confluent.io/blog/kafka-fastest-messaging-system/