Future Generation Computer Systems 134 (2022) 374–387

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Big data architecture for connected vehicles: Feedback and application examples from an automotive group

Ahmed Mostefaoui [a,*], Mohammed Amine Merzoug [a], Amir Haroun [b], Anthony Nassar [a,c], François Dessables [c]

[a] DISC Dept, FEMTO-ST Institute/CNRS UMR 6174, University of Bourgogne Franche-Comté, Belfort 90000, France
[b] Data & Connectivity Engineering Dept, Groupe PSA, Poissy 78300, France
[c] Data & Connectivity Engineering Dept, Groupe PSA, Bessoncourt 90160, France

A R T I C L E I N F O

Article history:
Received 2 December 2021
Received in revised form 25 March 2022
Accepted 19 April 2022
Available online 25 April 2022

Keywords:
Connected vehicles
V2I communication
Automotive big data
Big data architecture
Groupe PSA

A B S T R A C T

Nowadays, using their onboard built-in sensors and communication devices, connected vehicles (CVs) can perform numerous measurements (speed, temperature, fuel consumption, etc.) and transmit them, in a real-time fashion, to dedicated infrastructure, usually via 4G/5G wireless communications. This raises many opportunities to develop new innovative telematics services, including, among others, driver safety, customer experience, location-based services and infotainment. Indeed, it is expected that there will be roughly 2 billion connected cars by the end of 2025 on the world's roadways, where each of which can produce up to 30 terabytes of data per day. Managing this big automotive data, in real and batch modes, imposes tight constraints on the underlying data management platform. To contribute to this research area, in this paper, we report on a real, in-production automotive big data platform; specifically, the one deployed by Groupe PSA (a French car manufacturer known also as Peugeot-Citroën). In particular, we present the technologies and open-source products used within the different components of this CV platform to gather, store, process, and leverage big automotive data. The proposed architecture is then assessed through realistic experiments, and the obtained results are reported and analyzed. Finally, we also provide examples of deployed automotive applications and reveal the implementation details of one of them (an *eco-driving* service).

1. Introduction

In the V2I communication, connected vehicles, thanks to their various embedded sensors and telematics, can continuously feed their associated automotive infrastructure with large amounts of diverse useful data. For instance, at present, the connected vehicles of Groupe PSA[1] can report over 170 different types of data (vehicle identification number, GPS coordinates, revolutions per minute, etc.) and the number of these vehicles, which is already millions, is estimated to drastically increase by the next decade [1].

Such huge and continuously growing data offers interesting commercial opportunities and novel applications for the automotive manufacturer in question, its partners, vehicle owners, fleet managers, as well as different other customers. For example, automotive data can be leveraged to provide services to electric vehicles (e.g., battery pre-conditioning and charging), and can also be leveraged to improve the driving experience, road safety, fleet management, and vehicle services.

On top of this, and in contrast to the V2V paradigm, V2I also offers many other advantages (such as computation offload, data persistence, network coverage, and accessibility) and is more suitable for a broad range of applications, especially safety-related ones. For instance, while an early warning about the presence of freezing rain or black ice on a road network cannot rely on the V2V communication (due to the absence of information locality and persistence[2]), it can be effectively and efficiently handled by

* Corresponding author.
E-mail addresses: ahmed.mostefaoui@univ-fcomte.fr (A. Mostefaoui), amine.merzoug@gmail.com (M.A. Merzoug), amir.haroun@mpsa.com (A. Haroun), anthony.nassar@mpsa.com (A. Nassar), francois.dessables@mpsa.com (F. Dessables).

[1] Groupe PSA is the second-largest car manufacturer in Europe. It is present in 160 countries and possesses 16 production sites across the world. With 3.5 million vehicles sold worldwide in 2019 (https://www.groupe-psa.com/en/automotive-group/key-figures/).

[2] The alert remains alive as long as there is at least one vehicle in the area able to transmit the alert to the approaching vehicles, otherwise the alert information will be definitely lost.

https://doi.org/10.1016/j.future.2022.04.020
0167-739X/© 2022 Elsevier B.V. All rights reserved.

the V2I paradigm in which the infrastructure is responsible for ensuring data persistence.

Although the V2I paradigm offers numerous advantages and interesting commercial opportunities, it, however, requires the design, development, deployment, and operations of sophisticated platforms, in particular parallel and distributed ones, which are dedicated to big automotive data management (ingestion, storage, processing, leveraging, service offering, and data sharing). In the last couple of years, a lot of effort has been put into solving disparate big data challenges such as storage, processing, and leveraging. For example, as regards innovative big data storage and processing, extensive research has been conducted on this matter, and numerous efficient solutions and open-source software technologies have been proposed to deal with huge data volumes and make up for the deficiency and inadequacy of traditional techniques [1,2]. Also, as it pertains to big data leveraging through machine learning and advanced analytics, much research has and is still being done in this regard [3,4]. Despite all this work dedicated to dealing with big data storage, processing, and leveraging, to the extent of our knowledge, little attention has been devoted to the issue of proposing an end-to-end reference architecture for connected vehicles; a complete system or platform that can be in charge of managing big automotive data throughout its life cycle [1,2,5]: from the basic sensing operations to data ingestion, pre-processing, storage, processing, to the final exploitation phase (knowledge and insight extraction, data visualization, etc.).

To contribute to bridging the aforementioned gap, the present work reports on the big automotive data platform of Groupe PSA, which has been deployed and is in production for several years (Fig. 1). The goal and contributions of this work are fourfold: (a) Provide a detailed description of the layered big data architecture of Groupe PSA. (b) Narrate the state-of-the-art experience and feedback of this well-known car manufacturer group in terms of gathering, storing, processing, and leveraging its continuous streams of vehicle data. (c) Evaluate in a *thorough* and *realistic* manner this solution, and this in respect of three axes: overall performance, quality of service, and quality of data (i.e., data consistency, accuracy, completeness, and timeliness). (c) Provide examples of automotive application deployments with associated performances.

The remainder of this paper is organized in six parts as follows. Section 2 provides the main non-functional requirements that must be targeted when designing a big data platform for connected vehicles. Section 3 describes the big data architecture of Groupe PSA and details the layers and fundamental technologies of the deployed automotive platform. Section 4 presents the conducted experimental evaluations along with their results. The latter have been actually obtained (as previously mentioned) while considering/studying three dimensions: the overall *platform performance* and both *data* and *service* qualities. In Section 5, we present examples of automotive applications (namely; the weather and eco-driving services). We also provide the details of the implemented eco-driving application. Finally, Section 6 presents some related work and Section 7 concludes the paper.

2. Requirements for an efficient handling of automotive big data

An automotive architecture for big data gathering and leveraging must adhere to four major requirements: data privacy, time efficiency, high availability, and scalability.

- *Data privacy*: first and foremost, the proposed solution must provide customizable secure sharing. That is, vehicle owners must be able to share their data (e.g., with the manufacturer

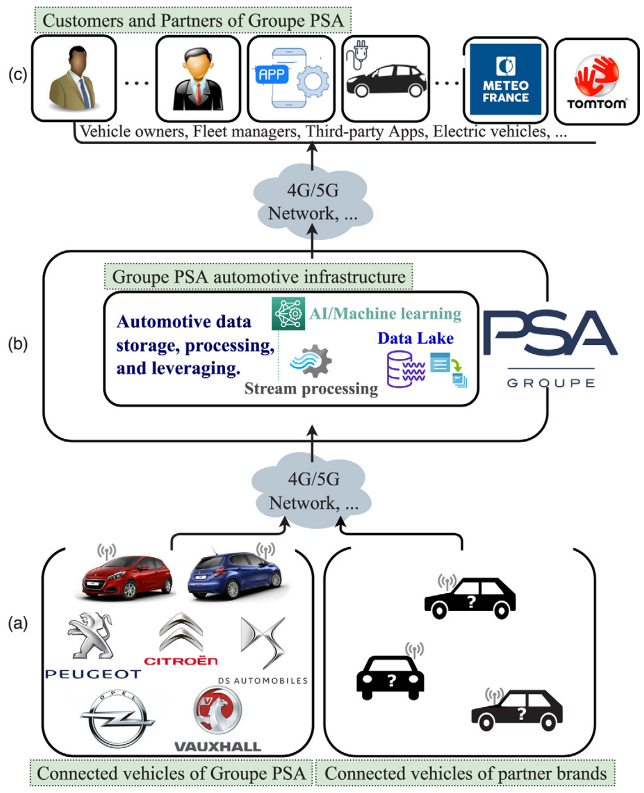

Customers and Partners of Groupe PSA
(c)
APP
METEO FRANCE
TOMTOM
Vehicle owners, Fleet managers, Third-party Apps, Electric vehicles, ...
4G/5G Network, ...
(b)
Groupe PSA automotive infrastructure
Automotive data storage, processing, and leveraging.
AI/Machine learning
Data Lake
Stream processing
PSA GROUPE
4G/5G Network, ...
(a)
PEUGEOT
CITROËN
DS AUTOMOBILES
VAUXHALL
Connected vehicles of Groupe PSA
Connected vehicles of partner brands

Fig. 1. Big data gathering and leveraging by Groupe PSA.

partners, third-party applications, and so on) while being able to control what information can be accessed and by whom. The implementation of this functionality must comply with the local laws and regulations. For example, across Europe, organizations and companies must comply with the GDPR[3]; the latest EU regulations on data protection and privacy. The latter imposes obligations onto organizations anywhere, as long as they collect and target data related to people in EU.
The collected automotive data is no exception and must comply with these rules. As any business with a digital online presence, it must be anonymized and filtered according to the customers' consents and preferences. To enable this, in the developed platform, clients must be offered contracts that list their preferences and can be accessed and modified at any time (e.g., anonymization parameters and partners that can access data).
- *Time efficiency*: to offer smooth services to final users, the proposed solution must use highly efficient storage and processing techniques (such as fast read/write operations). It must also support two mandatory modes; *online stream processing* and *data lakes* (also known as offline-batch storage and processing). The latter point (i.e., data lakes) will be detailed in Section 3.5.2.
- *High availability*: the proposed architecture must be highly robust, available, and fault-tolerant. In case of software or hardware failures, recovery must be instantaneous and transparent (without service interruption, and more importantly, without data loss). This can be achieved through numerous techniques such as data striping and redundancy across multiple cloud nodes, cloud-native applications with

3 General Data Protection Regulation: https://gdpr-info.eu/.

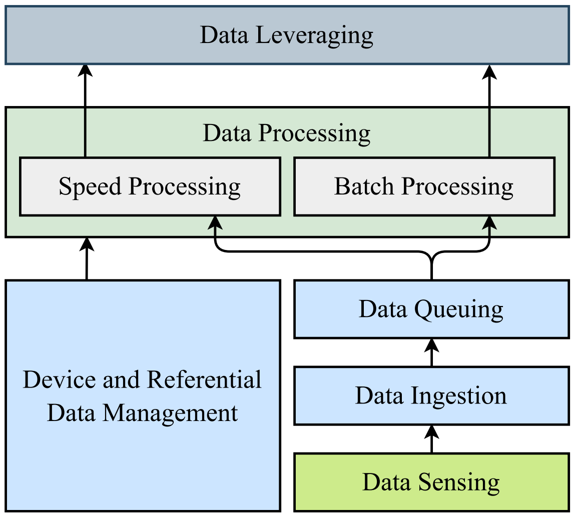


Fig. 2. Layered big data architecture of Groupe PSA.

respect to microservices and containers (e.g., dynamic management and self-healing in the cloud using for instance Kubernetes).

- *Scalability*: the proposed solution must be highly scalable so that it can maintain the required performance level in front of a large number of connected vehicles, their large diverse data, and the potential/inevitable increase of both. To do so, first, the different implemented or utilized out-of-the-box software components must be highly efficient in terms of data ingestion, storage, processing, and querying (both online and offline). This can be achieved by considering cutting-edge big data technologies and highly efficient state-of-the-art algorithms and approaches. Second, the proposed design must be able to change/grow over time without a wholesale replacement (i.e., scale-out by adding more components in parallel to spread out the workload, or scale-up by making a machine/component faster or more powerful so that it can handle the load at hand).

3. Big data architecture of groupe PSA

The current big data architecture adopted by Groupe PSA consists of six layers (as shown in Fig. 2): (1) data sensing, (2) data ingestion, (3) data queuing, (4) device and referential data management, (5) data processing (with two processing modes or sublayers), and finally (6) data leveraging and serving. We note that the first layer (i.e., data sensing), which is responsible for collecting data from the vehicles of PSA and those of its partner brands (cf. Fig. 1), is out of PSA platform. All the remaining layers, from (2) to (6), are implemented by the PSA platform (hybrid cloud, MQTT brokers, distributed producer–consumer streaming platform, database, data lake, etc.).

Before diving into the details of each of these layers, we point out that the policy of Groupe PSA focuses on utilizing efficient popular open-source technologies whenever they are available and suitable to serve the purpose.

3.1. Data sensing

The first layer in the architecture is related to sensing and communicating ambient environmental information from connected vehicles to a dedicated base station (Fig. 2). This essential step is realized mainly through the various embedded sensors

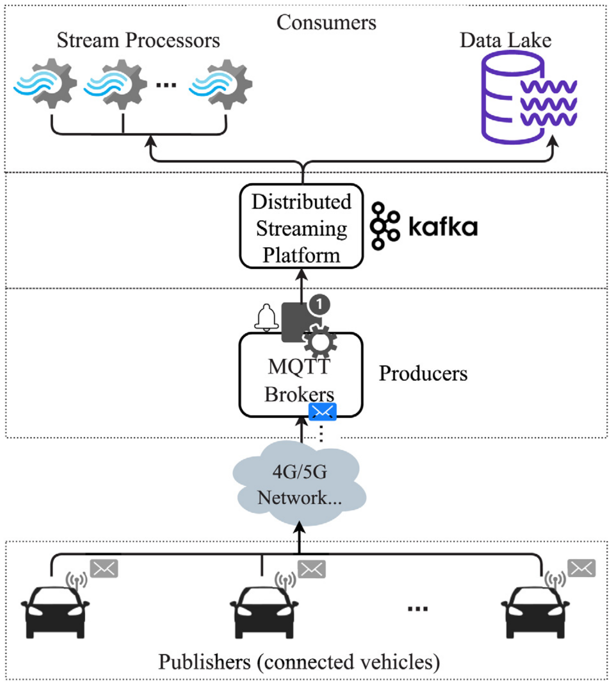


Fig. 3. Big data sensing, gathering, queuing, and processing.

and Telematic Service Units (TSUs) of vehicles, which capture automotive data streams and report them via mobile wireless networks.

Typically speaking, TSU units are installed in vehicles (connected to the CAN bus [6]) and have a data rate of 1 Mbit/s (according to the ISO 11898 standard [7]). Currently, Groupe PSA utilizes numerous kinds of TSUs, such as ATBs (Autonomous Telematic boxes), approved aftermarket boxes, and third-party telematic boxes. Each of these TSU units has a special application. For instance, units that are designed for extensive monitoring can be considered to send a large range of data at a very high frequency. As regards aftermarket TSUs, especially those that can provide features and functionalities (such as GPS localization) for low-end vehicles, they can be utilized to consolidate automotive data with additional information.

In addition to TSUs, V2X communication[4] can also be considered to send vehicles' data to smartphones and then to automotive sink. In both cases (TSU and V2X), data is sent in a binary format using the well-known lightweight MQTT protocol. As depicted in Fig. 3. This communication protocol connects, via data brokers, vehicles with their automotive infrastructure (publishers gather the required data and send it, through the brokers, to the sink).

3.2. Data ingestion

The role of this intermediate ingestion layer is twofold: (1) to ingest automotive data from the defined sources (i.e, sensing layer), and (2) to prepare it by making it usable by the upper layers. In other terms, whatever the communication protocol, the collected data must be ingested, decoded and formatted before it can be used. In the currently deployed platform, since the adopted protocol is MQTT, this front-end layer is composed of MQTT brokers responsible for receiving, decoding, formatting, and transmitting the output to the upper layer (Fig. 3). This

[4] V2X: vehicle-to-everything communication (Bluetooth, ...).

choice of protocol is motivated by numerous reasons. MQTT, which is one of the most widely adopted IoT protocols, has been specifically tailored to be simple and easy to implement (MQTT is IoT-oriented as it meets the strict requirements of embedded solutions). Moreover, this application-layer protocol has better performance than HTTP. Its lightweight features, including QoS controls, provide lightweight processing, fast communications, and allow devices to communicate with each other over unreliable, low-bandwidth networks [8].

3.3. Data queuing

To cover any unavailability of either the producers (e.g., ingestion layer) or consumers (e.g., stream processors and data lake), this layer acts as a *distributed asynchronous secure queue* with a predefined retention policy. This way, producers and consumers do not interact directly, messages will remain queued even after being retrieved by certain consumers, and all possible consumers can be resiliently and smoothly fed with the needed data. The previously utilized MQ (Message Queuing) or MOM (Message-Oriented Middleware) solution (such as IBM WebSphere MQ) suffered from several performance issues. For example, the proposed architecture was not fully distributed; it allowed the utilization of only one consumer, whether in the stream speed sub-layer or data lake. To remedy this, currently, the popular and widely utilized Apache Kafka[5] is used to implement the main tasks of this layer (Fig. 3).

As explained in the previous sections, the journey of data begins at the telematic boxes deployed in cars. On the backend side, where scalability is expected in addition to data retention, the strong point of MQTT is not the volume of messages that poses a problem, it is rather the ability to manage millions of connections from connected objects. Once data arrives at the MQTT brokers, the latter will take on the role of a load balancer managing a million TCP sessions in parallel and once data has been concentrated (gathered) in the brokers, it is then sent to the Kafka brokers (as the MQTT brokers have little storage capacity). In case of unexpected events between message production and consumption, Kafka will be used as a buffer. For example, if at one point a problem occurs at the telecom operator (the operator could not deliver the messages due to technical problems) then overnight a peak load of 500 million messages is sent, the MQTT broker must manage all connections for millions of devices and must quickly send messages to Kafka. The latter must in turn accept this load peak and above all store it so that behind, the consumers when they see this big wave coming, they cannot process at the usual speed but instead rely on the queue layer provided by Kafka.

Before addressing the processing layer which relies in an indispensable manner on the three previously described layers (data sensing, ingestion, and queuing), the next section will talk about the *device and referential data management* layer which is also very mandatory for the proper operation of the processing layer.

3.4. Device and referential data management

As shown in Fig. 2, the three basic (data sensing, ingestion, and queuing) layers and the referential one are independent. The former are responsible for automotive data gathering and preparing while the latter is in charge of managing devices (e.g., activation, deactivation, and update) and providing the necessary information required for the proper operation of the whole platform (it serves as a reference for the upper processing and leveraging layers).

[5] Kafka: https://kafka.apache.org/.

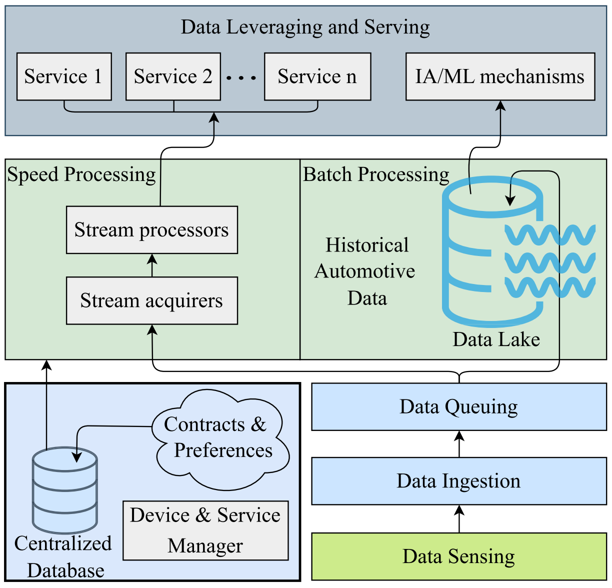


Fig. 4. Detailed big data managing architecture of Groupe PSA.

More technically speaking, all the necessary and crucial big data management functions and information (related to devices, vehicle owners, customers, partners, contracts, preferences, services, and so on) are delegated to this referential layer. For instance, it is responsible for defining access policies, i.e., who can access which resource (environmental, geographical, or other information) in which mode (B2B, B2C, or both). As a second example, the type of processing or even preprocessing, such as anonymization, filtering, and aggregation, that must be applied to the collected data are dictated at this level.

Regarding its implementation, this layer is composed of a device and service manager and a centralized database that stores all the mandatory information (Fig. 4). Vehicle owners, partners, and administrators are granted access to this database and can interact with it via their respective Web spaces.

3.5. Data processing

This layer, which is considered to be the most important component of the whole architecture, is divided into two fundamental sub-layers: speed processing and data lake (Fig. 4). The speed sublayer processes the unlimited data streams, and the data lake stores and processes historical data with offline jobs (such as machine and deep learning techniques).

Before detailing these two components, we recall that the collected automotive data is personal and sensitive and cannot be utilized or shared in its arrival raw state (it must be pre-processed before use). This applies to both data in motion and at rest. For example, an aggregation function on a particular set of vehicles or an average on a specific time-window must be performed before use.

3.5.1. Speed processing

The main tasks of the online processing sub-layer are accomplished through two communicating streaming microservices [9]; namely, data stream *acquirers* and *processors* (Fig. 4). The stream acquirers retrieve fresh automotive data from the lower queuing layer (Apache Kafka) and deliver it to the stream processors. The latter have one fundamental objective; processing the continuous data flows while (1) respecting strict (real or near real-time) constraints and (2) taking into account vehicle owners' preferences.

To do so, in addition to the data received from the acquirers, stream processors must also retrieve referential data from the reference layer (see Fig. 4).

Among the main technologies utilized at the level of this layer, we mention IBM Streams,[6] Apache Flink,[7] as well as different Kafka source and sink connectors, and Kafka producer and consumer APIs.

Next to the speed processing, we find the data lake, which provides efficient distributed *storage*, and also fast efficient offline *processing* for historical automotive data. To ease the reading and understanding of this upcoming part, it has been divided into two subsections instead of one: historical data storage (Section 3.5.2) and historical data processing (Section 3.5.3). After that, we will move to the last layer in the architecture (data leveraging and serving in Section 3.6).

3.5.2. Historical data storage

Several technologies can be utilized to store big historical data. The currently deployed platform employs: *data lakes* and *enterprise data warehouses* (EDW). By using such a hybrid unified system, the objective is to have an environment in which users can ask questions that can be answered by more data and more analytics with less effort. In the following, we provide a quick recall of EDWs and then data lakes.

EDWs were developed at most organizations to collect information from different sources so that reporting and analytics could be of use to everyone. EDWs support batch workloads and are designed for continuous use by hundreds to thousands of concurrent users who can conduct reporting or analytics tasks. It is however worth pointing out that the warehouse data model has to be carefully designed before loading data.

Indeed, data warehouses have been for some time the *ipso facto* solution for enterprises when it comes to storing data for later usage. But, when the big data wave arrived, most businesses, which invested plenty of money constructing their EDWs, have begun building data lakes, which can be seen as an improvement to what data warehouses were limited to do.

The concept of data lake was introduced in 2010 by James Dixon as follows [10]: "*If you think of a datamart as a store of bottled water – cleansed and packaged and structured for easy consumption – the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake and various users of the lake can come to examine, dive in, or take samples*". In other terms, a data lake is a vast data repository that is based on low-cost technologies and aims at enhancing the collection, refinement, archival, and mining of raw data within an enterprise.

The concept is simple; instead of putting data in a certain data store designed for specific purposes, just move it in its original format to the data lake. This technique reduces the upfront costs of data ingestion such as transformation. In other terms, raw data can be captured and stored at scale for a low cost. Once data has been deposited in the lake, it becomes available to all teams of the organization (which offers improved agility and accessibility for data analysis [11]). Among the other advantages and capabilities offered by data lakes, we also mention: storing many types of data in the same repository, defining the structure of data upon usage (schema-on-read), and the fact that a data lake includes the mess of raw unstructured or multi-structured data that, for the most part, has unrecognized value for the organization.

The implemented data lake and its interaction with the EDW are illustrated in Fig. 5. As the figure shows, the data lake is composed of two main components: the less adopted solution; the Hadoop system [12], and the current go-to solution; Object Storage [13]. It also encapsulates an Elasticsearch engine [14].

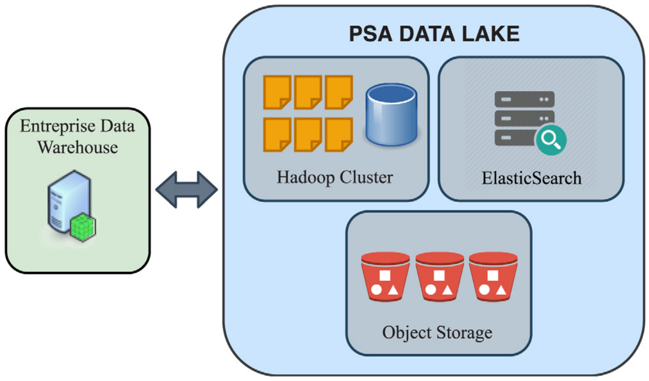


Fig. 5. Illustration of the data lake and EDW of Groupe PSA.

- *Elasticsearch*: is essentially a tool for indexing (but not exclusively for storing) data. Thus, it can be incorporated in a data lake to index its data. For example, if an organization or company had multiple storage systems in use to store raw source data, it is helpful for an application to consult a master index to decide which data resides within which system. Such implementation places Elasticsearch as a data catalog used to quickly find data across multiple storage systems. By doing so, moving data around will become simpler (e.g., copying data to HDFS for a MapReduce job, or directly reading from the source to load in memory to run in Apache Spark.[8]
- *HDFS*: in the early deployment of the platform, the only adopted storage was Apache Hadoop Distributed File System [12]. In a nutshell, this well-known solution has a master/slave cluster architecture (*NameNode* and *DataNodes*). While the NameNode manages the cluster and stores meta-data, DataNodes are responsible for data (or block) storage (files are split into blocks and then spread on different DataNodes). Despite its distributed nature and high-throughput access, HDFS suffered from some limitations like scalability and availability. For example, it did not allow independent scaling (computing and storage were tightly coupled; when a resource was added, the other had to follow). This limitation was a big issue in a hybrid cloud environment. Also, NameNode represented a SPOF (single point of failure) in the cluster. When it failed, the whole system crashed, and manual interventions were indispensable [12]. Currently, NameNode supports high availability (it is no longer a SPOF). As regards the scalability aspect, the current version of HDFS allows independent scaling of computing and storage. Indeed, since Hadoop 2.0 the compute part has been decoupled (YARN), hence allowing scaling compute and storage resources independently. Despite its advantages and solved issues, Hadoop still suffers from certain drawbacks such as the handling/management of small files. Since the metadata of all files is stored on the NameNode, this could make operations very slow with a large volume of files.
- *Object Storage*: from a technical perspective, Object Storage [13] is a solution that manages data as objects (an alternative to file and block storage, which manage data respectively as a hierarchy of files, and blocks within sectors and tracks). Object Storage is composed of extended metadata. The unique identifier attributed to each object allows servers to retrieve it from any physical location. Use cases of

[6] IBM-Streams: https://www.ibm.com/cloud/streaming-analytics.

[7] Flink: https://flink.apache.org/.

[8] Spark: https://spark.apache.org/.

Object Storage include cloud storage, photos, video, audio, and large image files.
Indeed, initially, Object storage was adopted/integrated in the platform as an ideal solution destined to overcome the old key drawbacks and limitations of HDFS mentioned above. Object Storage was not seen as a replacement for Hadoop (data can be easily accessed and processed on Hadoop when it is stored using Object Storage). It complemented Hadoop by offering a low-cost storage alternative and making processing more powerful.
Currently, Object Storage is still in use in the platform given its numerous advantages. For instance, Object Storage can easily scale out beyond Petabytes. Moreover, with Object Storage, data accessibility or data loss are prevented and secured through a data protection mechanism known as *erasure coding*. Object Storage offers simple Web service interfaces for access through RESTful APIs. Another motivation of using Object Storage is the security aspect. At present, it is difficult to handle the security in HDFS. One has to use Kerberos[9] for authentication and Apache Ranger[10] for authorization and encryption.

3.5.3. Historical data processing

At present, this part of the architecture is implemented using Apache Spark,[8] which is one the well-known and most utilized frameworks for batch processing. In brief, this tool has numerous advantages, among which we cite that it has been proven to be very efficient and adequate in terms of multi-pass computation such as iterative machine learning.[8] Spark offers ease of use and development through different programming languages such as Java, Scala, Python, and R. Spark is very fast on memory and provides a fault-tolerant distributed memory abstraction via its implementation of Resilient Distributed Datasets (RDDs) [15]. Finally, the benefit of using Spark is that it does not need to write data to disk between stages contrary to MapReduce [16] (which was originally designed for commodity hardware where persisting data was a requirement).

3.6. Data leveraging and serving

This component takes the data produced by the lower data lake and speed layer, indexes it, and serves query results to final users. In the following, we first talk about data indexing and then data serving.

A layer of virtualization is needed to simplify the accessibility of data (handling its underlying complexity). In [17], *Rick F. van der Lans* defined data virtualization as "*the technology that offers data consumers a unified, abstracted, and encapsulated view for querying and manipulating data stored in a heterogeneous set of data stores*". The data virtualization software aggregates structured and unstructured data sources through a dashboard (or visualization tool) for virtual viewing. Such software allows for the discovery of metadata about data but hides the complexities associated with accessing disparate data types from different sources. In other words, data virtualization does not replicate data from source systems; it only stores metadata and logic for display integration.

To index data, two NoSQL databases must be considered: batch and real-time random read/write queries [18]. For instance, in the currently deployed architecture, it has been opted for Apache HBase[11] and Apache Phoenix.[12] HBase is a distributed column-oriented NoSQL database modeled after Google BigTable [19]. As for Phoenix, it is an open-source SQL engine that provides *low-latency* queries for data stored in HBase. In this regard, we briefly recall that NoSQL might also refer to Not Only SQL (to indicate that it can also support SQL-like query languages) [20]. NoSQL databases have four basic types (key–value, column, graph, and document) and three common characteristics [18]: high scalability, data replication, and schema-less (no schema needs to be followed). The objective is to deal with large increasing data size and increase the system reliability in front of losses.

Now, as regards service offering to end-users, in the current deployment, the leveraged automotive data is exposed in two modes: MQTT and RESTful APIs [21,22]. In the same context, to grant third-party clients (such as mobile applications) access to the collected personal automotive data without exposing users' credentials, the open-standard authorization protocol OAuth has been considered. Finally, as depicted in Fig. 4, this data leveraging and serving layer also applies multiple artificial intelligence and machine learning techniques to make the data at hand (stored in the data lake) relevant to numerous questions and problems of interest (such as improving new products/cars, improving customer services and all sorts of activities).

4. Experimental evaluations

In this section, we evaluate the deployed solution according to two main criteria: performance and quality.

- The *performance evaluation* translates into examining the technical behavior of the platform when facing data flow fluctuations (i.e., platform scalability). In this axis, we will observe how Groupe PSA uses a reactive policy to deal with trends accompanying the rising number of connected vehicles (Section 4.2).
- The *quality dimension* consists of dealing with two sub-characteristics [23]: *quality of data* and *quality of service* (Section 4.3 and Section 4.4). The aim is to support and justify the technological choices that have been made to implement the different components of the architecture.

Before talking about the performed tests and obtained results, the next Section 4.1 will present the hardware settings on top of which the described architecture is deployed.

4.1. Underlying hardware

The big data infrastructure of Groupe PSA spans onto two data centers: DC-A and DC-B, which contain respectively 80 and 50 *nodes*. Each data center is composed of *Management Nodes* (MN) and *Data Nodes* (DN) (see Table 1 for technical characteristics). The two data centers, distant about 700 km from each other, are linked through private Gib-Ethernet communication links.

Big data applications are implemented in both data centers and replicated accordingly in each. A Hadoop ecosystem distribution is installed in both. The applications range from storage to data processing platforms such as GPFS, HBase, Phoenix, Spark, and Machine Learning platforms. This infrastructure mainly handles IoT data sourcing from Connected Vehicles and the Factory of the Future.

4.2. Performance evaluation

Today, the automotive sector of Groupe PSA accounts for one *million* monthly active connected vehicles (users) in France only. This number represents the CVs whose owners have agreed to transmit their data to the infrastructure. Thus, these CVs are perpetually transmitting around 65 *million messages* per day on average.

9 Kerberos: https://www.kerberos.org/software.
10 Ranger: https://ranger.apache.org/.
11 HBase: https://hbase.apache.org/.
12 Phoenix: https://phoenix.apache.org/.

Table 1
Big data infrastructure of Groupe PSA.

	Data center A # nodes	Data center B # nodes	Technical characteristics
Management nodes (MNs)	6	6	32 Cores (64 vCPUs). 256 GB RAM. 2x 10 Gib/s Ethernet adaptors. 8x 600 GB HDD SAS 15000 rpm.
Data nodes (DNs)	74	44	32 Cores (64 vCPUs). 256 GB RAM. 2x 10 Gib/s Ethernet adaptors. 10x 6TB HDD SATA 7200 rpm.

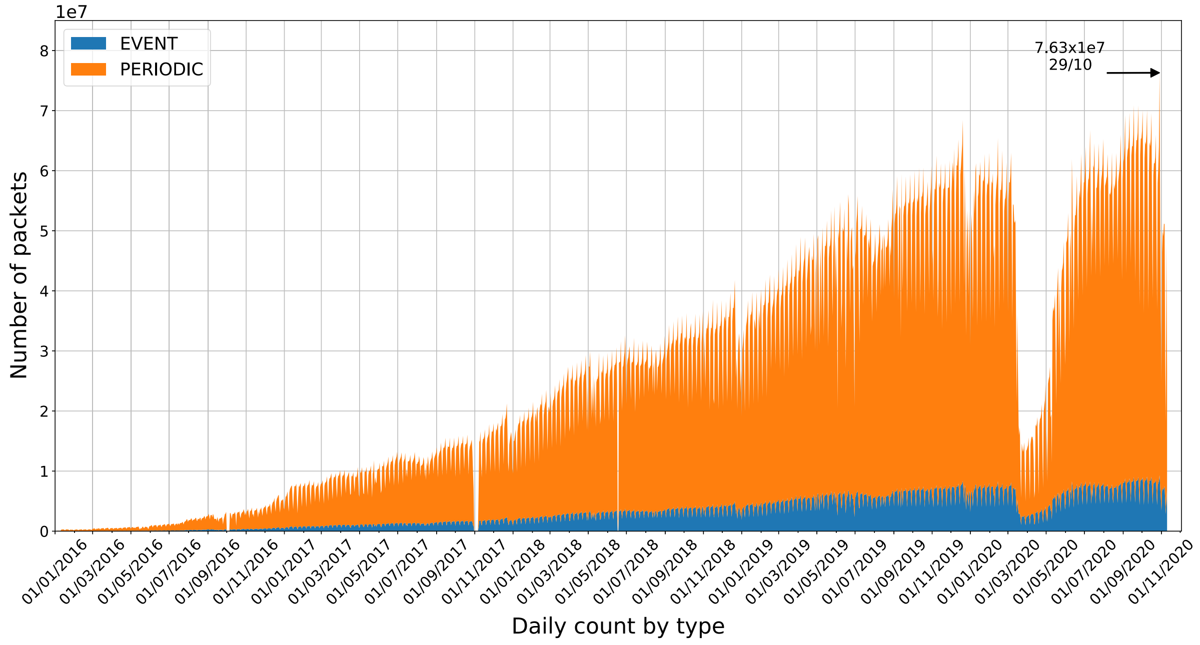


Fig. 6. Evolution of transmitted CV packets throughout the years 2016 to 2020.

Fig. 6 shows the evolution of the number of sent CV packets throughout recent years (2016–2020). The blue segment/layer on the bottom corresponds to *eventual packets*, and the orange one corresponds to *periodic packets*. These two types of packets are complementary. The periodic packets are recurrent sensor measurements embedded in the vehicles. Whereas, the eventual ones are generated following the occurrence of a set of predefined events, such as starting or stopping car engines, crash incidents, technical alerts, or even when turning on/off the privacy mode.

The figure highlights a significant decrease in the activity of CVs during the period of March–May 2020, which corresponds to the COVID-19 confinement period in France. The figure also shows an activity peak (corresponding to 76 *million* messages) due to people rushing to stores during the beginning of the second partial confinement of November 2020. As regards the abrupt discontinuities or gaps in the graph, they are actually due to occasional system upgrades or maintenance downtime. These irregularities decrease in time thanks to the implementation of redundancy policies.

Like many other organizations, Groupe PSA must deliver business continuity. To achieve this end, disaster recovery plans have been implemented through multi-data center redundancy with servers and other critical infrastructure at a different location to that of their primary site. Moreover, when system updates become available, all cluster machines are upgraded in a *rolling update* fashion. This way, servers update one after the other without introducing any disruption to the offered services.

Fig. 7 shows the variation of the number of transmitted CV packets in a period of two days. Every day, we observe approximately the same pattern with more or fewer data. Throughout the day, the generated data is not constant because vehicle usage varies and peaks in the morning and night. These peaks correspond respectively to people's movements when they leave for work and when returning from it. In this second scenario, we observe a significant change of load on the infrastructure. Though it is not sudden, it is a flow that remains variable and could change, especially in pandemic times where people either use more or less their cars depending, for instance, on whether there is a lock-down period approaching. Likewise, on weekends and holidays, the transmission frequency is also altered.

Before moving to the next point, we point out that the colored segments/layers in Fig. 7 (the stacked area plot) represent the content of 24-partition Kafka topics. Currently, the Kafka production environment is composed of 9 nodes. We configured topics of 24-partitions with a replication factor of 3 (i.e., 72 partitions in total). We also point out that in all the previous scenarios, the architecture handled the fluctuations intelligibly. It remained available to deliver with no registered downtime. This is mainly due to the distributed frameworks used on all levels of the architecture, whether speed processing layer (e.g., IBM Streams and Apache Flink), queuing (temporary storage) layer (i.e., Apache Kafka), or batch processing (e.g., Apache Spark).

The second performance plan consists of measuring the upper performance of the architecture; i.e., to what extent it can handle the load of Big Data applications. To do so, the idea would consist

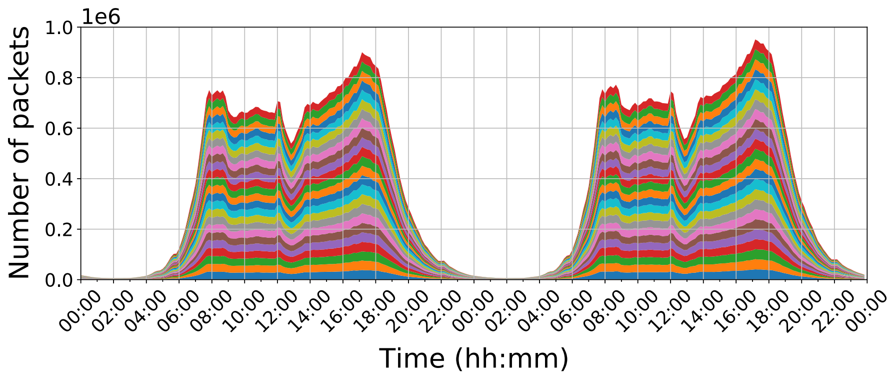


Fig. 7. Number of CV packets transmitted over a period of two days.

of saturating/stressing the platform with incoming messages and observe its behavior. Nevertheless, as the targeted platform is in production, we were not allowed to completely saturate it (given the high risk of shutting down the whole system). Therefore, we performed our tests and measurements during a *low activity period*; typically between 1 am and 3 am (as it was shown in Fig. 7).

More specifically, we gradually increased the number of messages sent to the platform, starting from the highest previously observed value (i.e., 950K messages in 5 min interval, which corresponds to 3100 messages/second as shown in Fig. 7). We then increased this number to 95% of the platform saturation. We reached 19511 messages/second and hence found out that the proposed architecture, mapped on the current infrastructure, can handle up to 7 *times* the actual peak load. This result validates the effectiveness and efficiency of the proposed architecture.

4.3. Quality of data

Data quality is made up of many dimensions that can be grouped into two groups [24]: (1) *intrinsic*; referring to attributes that are objective and native to data and (2) *contextual*; referring to attributes that depend on the context in which data is observed or used. Relevance, added-value, quantity, credibility, accessibility, and reputation of data are qualitative measurements. Self-report surveys and user questionnaires have relied heavily on these dimensions indicators since they rely on decision-makers' subjective and situational assessments for quantification. Therefore, contextual dimensions tend to deal with information rather than data since these dimensions are built when data is put in a situation or issue-specific context. Since we consider data quality, not information, we restrict our quality discussion on *intrinsic data quality* measurement.

Data quality is evaluated around two phases of the big data value chain [23]: at data reception and downstream after processing. In this paper, we focus on data quality at the pre-processing level, i.e., at the gathering layer after transmission from vehicles to infrastructure. Actually, the pre-processing in this architecture can be divided into two steps: (1) the decoding of raw binary data arriving from telematic boxes, and (2) the pre-processing happening at a later stage (at the beginning of the stream processing chain filtering inconsistent data).

Intrinsic data quality measurement is defined in four dimensions [25]: consistency, accuracy, completeness, and timeliness.

- *Consistency:* inspects the level to which related data records remain consistent with the predefined structure and format.
- *Accuracy:* this metric is related to the level of how well the reported information is accurate and resembles the real-world values and is therefore reliable.
- *Completeness:* concerns the level to which records would be considered complete in content (with no missing data).
- *Timeliness:* inspects the extent to which data can be considered as up-to-date.

Table 2
Deserialization results — error exceptions.

Exception type	Percentage
`DecoderException`	97.47%
`DateTimeException`	1.29%
`IllegalArgumentException`	1.12%
`BufferUnderflowException`	0.13%

In the following, we report the obtained experimental results concerning these four metrics.

4.3.1. Consistency

Data consistency is calculated according to the following formula [23]:

$$CNSM_a = (numInconsistentRecords \;/\; totalRecords) \qquad (1)$$

This criterion represents the ratio between the number of inconsistent data records to the total number of data records. In a typical 24 hour period, the infrastructure of Groupe PSA receives 65 million packets, among which 130k raise an exception when processed. Therefore, the *error rate* is approximately 0.2%. This result is very encouraging and confirms the effective performance of the proposed architecture.

The errors raised by the platform, in particular the exceptions raised by the deserialization of binary data coming from vehicles, are summarized in Table 2. As depicted by the reported results, we can see that four types of exceptions are encountered and that the majority originate from the *DecoderException* with 97.47%. The other three exceptions account for less than 3%. The reason behind this difference is that the *DecoderException* class is the most generic exception; so, errors are more prone to falling under this category.

4.3.2. Accuracy

Regarding the content accuracy, we used the following formula [23]:

$$AM_a = (numCorrectValues \;/\; totalValues) \qquad (2)$$

This metric summarizes the ratio between the number of records correctly received to the total number of records. The obtained results show that 99.9% of data was correctly received. This low satisfying number of errors related to the correctness of records is actually due to the fact that the first serialization step filters almost all aberrant packets.

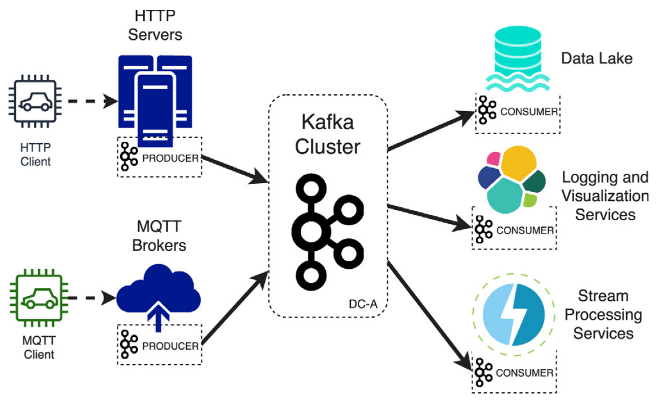


Fig. 8. Servers and applications using Apache Kafka as a messaging system for data transmission between layers.

4.3.3. *Completeness*

Here, the obtained results confirm that 100% of transmitted data was complete with no missing values. This demonstrates that the proposed architecture ensures data completeness.

4.3.4. *Timeliness*

In the current running configuration, there is an update every one minute (during which each connected vehicle sends a packet). The platform can process (i.e., receive, decode, clean, and store) this CV data in less than one minute (typically between 2 and 25 s). Therefore, we can say that the proposed automotive platform fully respects this data metric.

4.4. *Quality of service: messaging system evaluation*

To evaluate the QoS, we chose to assess the communication layer that connects all the architecture components. In fact, as in any typical big data architecture, the different components/processes work together (to collect, clean, process, and persist data) and thus require a robust, fault-tolerant, and ubiquitous platform/heart, which plays the role of a messaging system able to reliably and efficiently dispatch all the needed data throughout the whole architecture.

In the following, we evaluate Apache Kafka[5] (the employed messaging system) by testing its performance on different levels. We recall, as shown in Fig. 8, that Kafka is utilized to connect several clusters of machines, each dedicated to a particular task (i.e., stream processing, batch processing, storage, and visualization platforms). We also mention that the fundamental element of Kafka is the client instances (both *producers* and *consumers*) running on these clusters.

In the conducted evaluation, we considered data transmission performance from the producer and consumer clients (running on different systems) to the Kafka cluster/brokers. In more exact words, we analyzed the *throughput* and *latency* from Kafka clients to Kafka brokers. The services for which the communication client–server was tested are: (1) the HTTP servers responsible for decoding the transmitted data, (2) streaming servers holding stream processing applications, and (3) data visualization services, which help decision-makers to get better insights about their products.

To study the impact of deploying an application on a particular data center, we ran the corresponding Kafka clients (consumer or producer) of the above services on two separate data centers: DC-A and DC-B. We did not impose any throughput constraints so that the Kafka client can take full advantage of the infrastructure capabilities. We set the size of messages to be 1 KB (CV

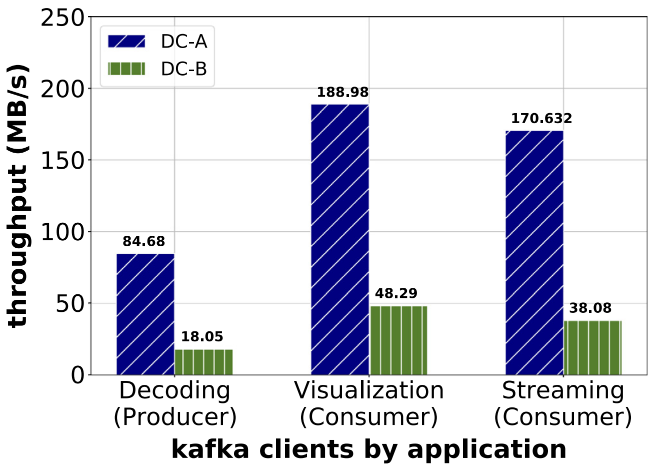


Fig. 9. Kafka client throughput vs. servers/applications.

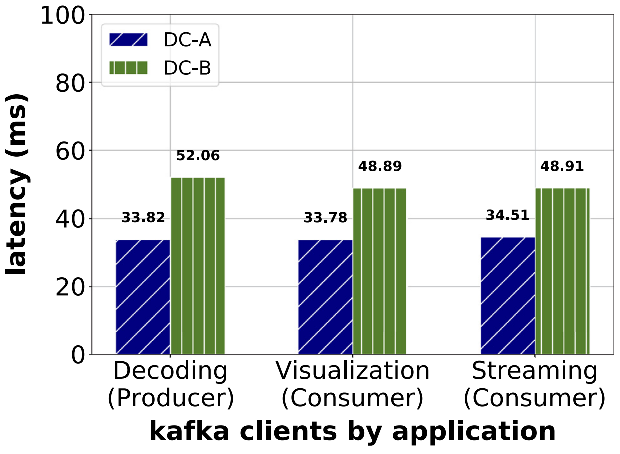


Fig. 10. Kafka client latency vs. servers/applications.

application). The Kafka brokers were deployed in DC-A. Figs. 9 and 10 show respectively the throughput and latency of several applications to the Kafka brokers. As expected, co-locality (i.e., the fact that DC-A holds the Kafka clients) led to higher throughput and lower latency.

Another issue we considered is measuring the impact of compression on the platform throughput. To this end, we used and compared different available compression algorithms (namely; Zstandard, LZ4, gzip, and Snappy). Fig. 11 presents the obtained results and shows that the best performance was obtained when using the Snappy compression algorithm.

In conclusion, we point out that, in the production environment, applications and platforms that need high throughput and low-latency data transmission, such as stream processing, were deployed in the data center closest to where the data is stored. However, as regards the other applications such as batch processing and visualization tools (which build graphs about daily, monthly, or yearly data), they are/can be placed separately from data.

5. Deployment examples of automotive applications

In this section, we describe two services that are presently deployed via the automotive platform of Groupe PSA: the *weather* and *eco-driving* services. We also provide the implementation details of the latter.

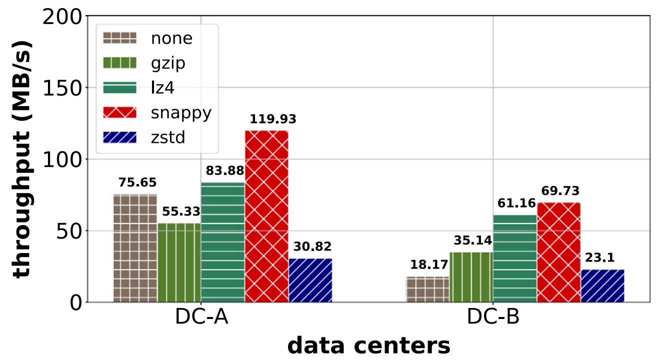


Fig. 11. Load testing of the messaging system by transmitting messages to different data centers using different compression algorithms.

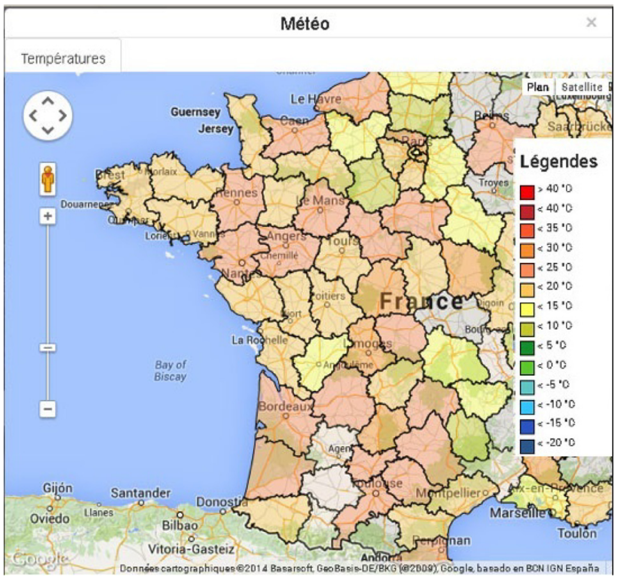


Fig. 12. Weather service.

5.1. Weather service

To meet the requirements of this application (which is in partnership with Météo–France[13]), each connected vehicle captures the external ambient temperature, tags it with its current location, and then sends the result to the infrastructure. The latter correlates the received data and updates the map (Fig. 12). The goal is to provide a precise real-time overview of the whole country's temperature through an updated map that displays the different regions and departments.

5.2. Eco-driving

The goal of this service is to advise drivers and help them improve their driving style (making it more eco-friendly). The current assessment of the customer driving style, adopted by Groupe PSA, is represented by scores. The global eco-driving score and its sub-scores are divided into two types: (1) the driving style; a score on the driver's behavior, and (2) the road profile; a score related to the selected itinerary. In other terms, this application aims to accurately assess and analyze the driving style of customers by taking into account their fuel consumption, acceleration, speed, braking, RPM, etc., and correlating them with other relevant environmental data (such as altitude, road's slope, and other customers driving information).

This eco-driving service is implemented as a stream processing job in the IBM Streams processing engine[6] (i.e., it is implemented in the speed processing layer of Fig. 4) [26]. More concretely speaking, this service is composed of several Processing Elements (PEs[14]) deployed on hosts that can communicate with each other either directly via shared local memory or through a network (depending on the adopted deployment and fusion strategy). Fig. 13 visualizes a simplified version of the application's data-flow graph in which are exhibited the main operators and their relationships.

The complete graph of the developed application is depicted in Fig. 14. Notice that this application consists of three regions (ingestion, parallel, and dissemination) and that the heavy work is done in the parallel one. Indeed, since the required processing operations are independent and consume the same data as input, they can be executed in a parallel fashion (to speed up processing). At the end of the parallel region, there is a barrier that aims to block the flow until all results are ready to be grouped and forwarded downstream.

This application graph (presented in Figs. 13 and 14), needs to be *mapped* on the underlying physical platform (i.e., the necessary computing resources must be allocated to each of its PEs). This process is actually governed by what is known as the *fusion strategy*, which consists of finding the optimal mapping scheme so that the job as a whole can process high rates of data.

Usually, this important task is left to the underlying software (in our case, IBM Streams), which generates a *standard* mapping based on paralleling PEs. However, in previous work [27], we have studied this issue and demonstrated that the standard/automatic approach does not yield the best performances in the context of connected vehicles. To remedy this, we have proposed and implemented a new approach based on integrating infrastructure features, mainly communication links features, to improve the overall application performance. For instance, in Fig. 15, we plot the execution time of the eco-driving application with both the straightforward fusion approach (called *distributed* in the figure) and with our proposed approach (called *merged*). The observed improvement is about 4 %, which, when translated in the current deployed CV platform, means that more than 800,000 additional vehicles can be handled.

6. Related work

Connected vehicles, with all their facets, have been a hot topic in recent years. Despite this fact, the literature review has highlighted the lack of platforms and architectures that are destined to ingesting, storing, and processing the big data generated by connected vehicles. The following paragraphs present some works that we deem related to the topic addressed in this paper.

One of the earliest works that has addressed the topic of data sharing in telematics was presented by Duri et al. [3]. This work defined some telematic services and policies to share and access data of connected vehicles. For example, to minimize the interaction with drivers, the authors have proposed a flexible privacy policy engine that allows data owners to specify access policies to their personal data. The proposed privacy model does not only determine who can access data, but also for what purpose, under what constraints, for how long data can be kept, and to whom this data can be distributed. Among the drawbacks of this

[13] French National Meteorological Service: https://meteofrance.com/.

[14] Generally speaking, PEs are the execution units created after compiling a stream processing application.

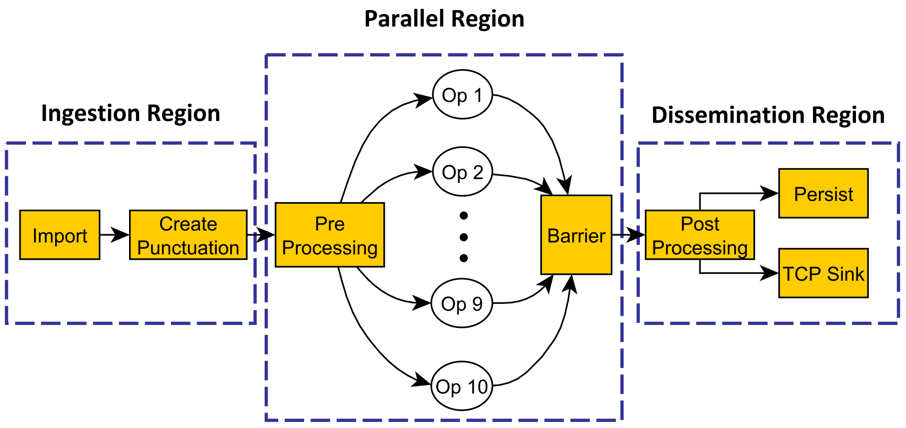


Fig. 13. Illustrative data-flow graph of the eco-driving application.

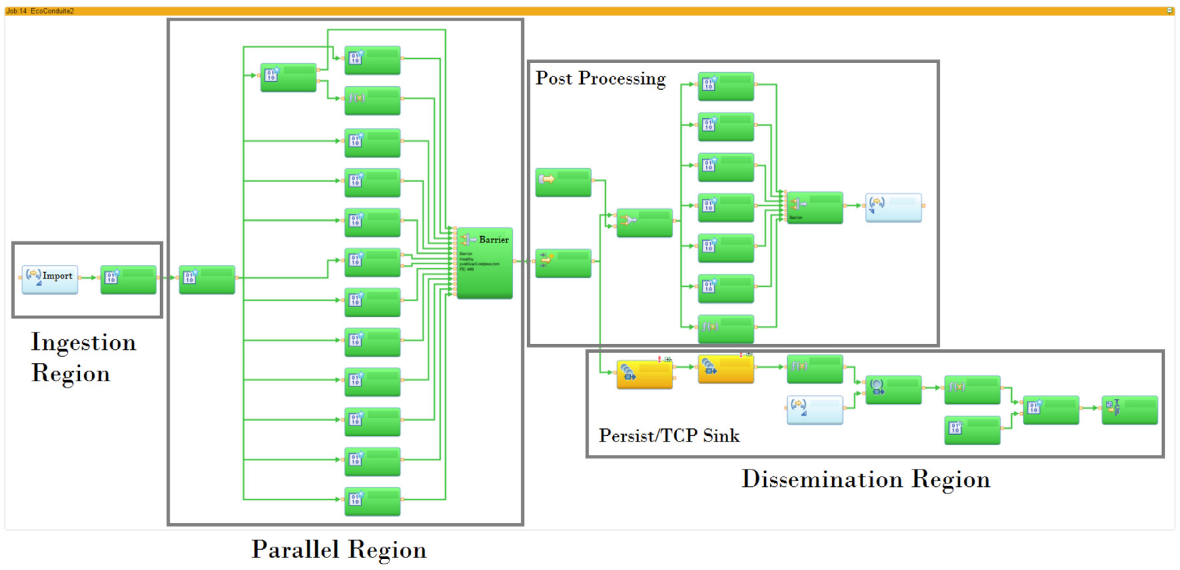


Fig. 14. Data-flow graph of the developed application (screen-shot from the IBM Streams Studio IDE).

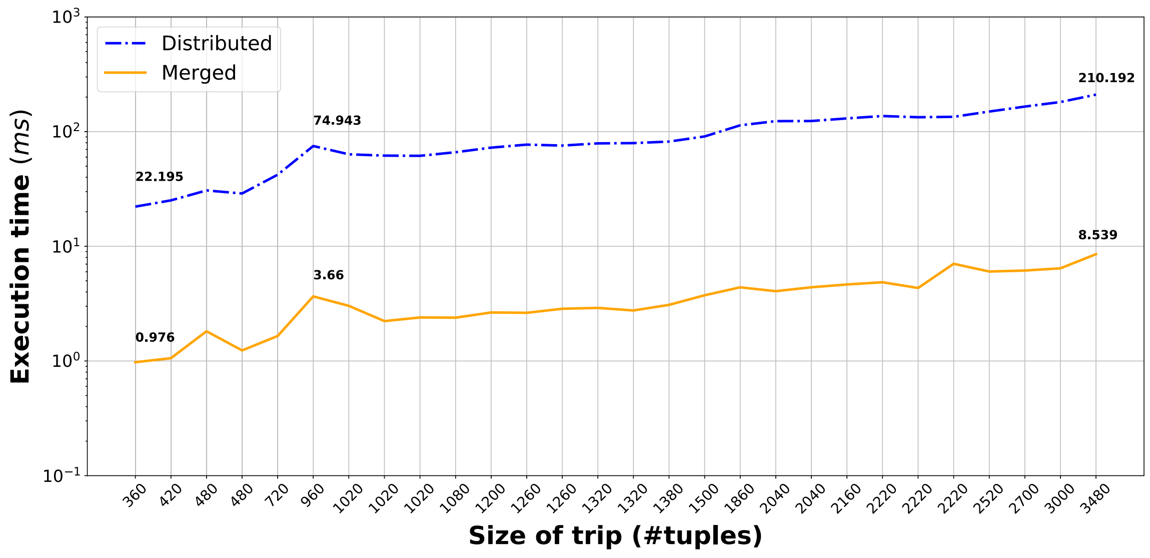


Fig. 15. Merged versus distributed fusion strategies (over 28 trips).

solution/framework are its complexity and poor performance, especially when compared with the big data architecture presented in this paper.

In [4], Johanson et al. presented BAuD; a big automotive data framework that was designed to capture and analyze online data of connected vehicles. The goal is to be able to cover automotive needs such as extensive CAN bus monitoring, remote diagnostic read-out, and vehicle state-of-health (SOH). More in-depth, this BAuD framework has six core components: (1) telematics service platform, (2) cloud-based back-end infrastructure, (3) task manager, (4) data broker, (5) analytics service architecture, and (6) web-based user interface (front-end). First, data is sent from the telematics service platform as Measurement Data Format (MDF) and stored on a distributed file system. Then, this data is analyzed using the analytic framework (based on an analytics task definition). The web-based user interface is used as an interacting tool with the framework. Even though this framework responds well to some automotive needs, it, however, does not cover all the automotive requirements such as real-time analysis and data sharing.

Another well-known general-purpose big data solution is the Lambda Architecture proposed by Warren and Marz in [2]. This generic scalable big data architecture was designed to handle massive volumes of both real-time (low-latency) and historical (high-latency) data. To do so, the Lambda Architecture was designed to be composed of three main parts: *speed*, *batch*, and *serving* layers. The batch layer has two main functions: (1) storing the master data-set as an immutable append-only set of raw data and (2) pre-computing the batch views. The speed layer computes low-latency speed views (it compensates for the high-latency of the batch layer). And finally, as regards the serving layer, it indexes the data stored on the batch layer so that it can be efficiently queried. Despite the numerous advantages of this architecture, it is however very difficult to keep the processing treatments perfectly synchronized on these two complex and different speed and batch frameworks [28]. In other words, any modification in one layer must be reported to the other layer, otherwise, numerous issues will arise during the merging of their corresponding results.

The Kappa architecture was introduced in a blog post by Jay Kreps[15] (one of the original authors of Kafka and a data architect at LinkedIn at the time). In this post, Jay Kreps gave his opinion about the Lambda architecture and proposed a simpler alternative architecture called Kappa. In particular, the author did not agree that real-time processing was inherently approximate, less powerful, and more lossy than batch processing. Indeed, the author agreed that the existing stream processing frameworks are less mature than batch processing ones (e.g., MapReduce), but there is no reason that a stream processing system cannot give as strong a semantic guarantee as a batch system. Also, the main disadvantage of the Lambda architecture, according to the author, is the presence of two different subsystems (batch and speed layers) requiring two different code bases that must be maintained and kept in sync so that processed data produces the same result from both paths. Moreover, the subsystems themselves have to be maintained. This implies dealing with a certain number of specialized tools belonging to each ecosystem.

The idea of the Kappa architecture is to remove the batch layer and process data with only a stream processing system (equivalent to that of the speed layer), which can do reprocessing whenever needed. Hence, the architecture is composed of only two layers: speed and serving layers. The latter, has the same role as in the Lambda architecture. To implement reprocessing in the speed layer, a full log of the data to reprocess needs to be retained. When a reprocessing is required, a new instance of a stream processing job starts processing at the beginning of the retained data. This way, the same functionalities of the Lambda Architecture are provided, but with fewer frameworks. As regards new data, it is processed by the speed layer, which also (1) generates the views that will be stored in the serving layer and used for computing the result of the queries made by the applications, and (2) stores data that needs to be reprocessed. To summarize, the real advantage of the Kappa architecture is not related to efficiency, but rather allowing the development, test, debugging, and operations using a single processing framework.

In [5], the authors detailed two cloud-based, scalable IoT back-end frameworks/solutions built for processing vehicular data in various use case scenarios (such as CAN data collection, remote device flashing, eco-driving functionalities, and weather report/forecast). The authors also presented the evolution of the proposed solution as well as the major problems faced during the development of such an IoT platform for receiving, managing, distributing, and visualizing vehicular data in the considered scenarios. Specifically, the authors proposed two reference cloud implementations. The first one is an IaaS-based system (with a reference implementation deployed on an OpenNebula based cloud), whereas the second is an industrial PaaS-based solution (built on the Cloud Foundry platform within the premises of an automotive supplier company). Both versions were evaluated and validated with benchmarks.

Some other early works related to cloud-based solutions for connected vehicles are the ones proposed in [29] (developing vehicular data cloud services in the IoT environment) and [30] (the connected car in the cloud: a platform for prototyping telematics services). In the literature, we also find numerous big data solutions (platforms and architectures) related to smart cities and other specific domains. Among these solutions, we mention: PROMENADE [31](Graph-based general-purpose big data platform for smart-cities), DIMMER [32] (City energy efficiency), InterCity [33] (Parking application), Snap4City [34] (Mobility and transport), Sipresk [35] (ITS and traffic congestion), BRT [36] (Transportation planning), CEFIoT [37] (Surveillance camera system), NIMBLE [38] (Industry 4.0), Druid [39] (Exploring log data), RADStack [40] (Online advertising, based on [39]), SmartSantander [41] (Monitoring and irrigation), CiDAP [42] (Monitoring, based on [41]), Scallop4SC [43] (Smart houses).

To conclude this section, we state that the big data architecture of Groupe PSA was initially and mainly inspired by the Lambda Architecture. However, since then, numerous improvements, new features, and important technical refinements have been introduced. For example, the high-latency applications in the batch layer have been separated from those requiring low-latency in the speed layer. This way, the need to maintain the speed and batch layers synchronized has been eliminated.

7. Conclusion

This paper presented and evaluated a real in-production platform that is concerned with managing big automotive data throughout its life cycle; from data sensing, gathering, storing, and processing to the final leveraging step. Although numerous researches have been done and still undergoing on efficient big data storage, processing, and leveraging, to the best of our knowledge, not much attention has been given to the aspects tackled in this work.

To bridge this gap, we have rigorously described how Groupe PSA (the second largest car manufacturer in Europe) gathers, stores, processes, and leverages its big CV data. These different tasks, along with the technologies that ensure them, were presented in a structured manner using the actual deployed big data

[15] Questioning the Lambda Architecture: https://www.oreilly.com/radar/questioning-the-lambda-architecture/.

managing architecture. We have also experimentally and thoroughly evaluated the proposed architecture while considering three dimensions: overall performance, quality of service, and intrinsic data quality (i.e., consistency, accuracy, completeness, and timeliness of data). The obtained results (presented in the paper) confirmed the effectiveness and efficiency of the proposed architecture in all the considered dimensions.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Anthony Nassar reports financial support was provided by ANRT (Association Nationale de la Recherche et de la Technologie), France.

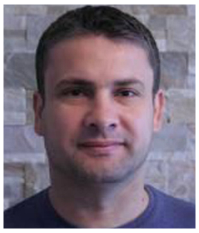
Acknowledgment

The authors would like to thank the Engineering Team at Groupe PSA for their precious contribution to this work. Technical contribution was also brought to this work under the CIFRE project (N 2017/0682) within Groupe PSA, funded by the French Ministry of Higher Education and Research and managed by the ANRT (Association Nationale de la Recherche et de la Technologie).

References

[1] A. Haroun, A. Mostefaoui, F. Dessables, A big data architecture for automotive applications: Psa group deployment experience, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, IEEE, 2017, pp. 921–928.
[2] J. Warren, N. Marz, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, Simon and Schuster, 2015.
[3] S. Duri, J. Elliott, M. Gruteser, X. Liu, P. Moskowitz, R. Perez, M. Singh, J.-M. Tang, Data protection and data sharing in telematics, Mob. Netw. Appl. 9 (6) (2004) 693–701.
[4] M. Johanson, S. Belenki, J. Jalminger, M. Fant, M. Gjertz, Big automotive data: Leveraging large volumes of data for knowledge-driven product development, in: 2014 IEEE International Conference on Big Data, Big Data, IEEE, 2014, pp. 736–741.
[5] A.C. Marosi, R. Lovas, Á. Kisari, E. Simonyi, A novel IoT platform for the era of connected cars, in: 2018 IEEE International Conference on Future IoT Technologies, Future IoT, IEEE, 2018, pp. 1–11.
[6] M. Di Natale, H. Zeng, P. Giusto, A. Ghosal, UnderstandIng and using the Controller Area Network Communication Protocol: Theory and Practice, Springer Science & Business Media, 2012.
[7] I.S.O. 11898-1:2015, Road vehicles - controller area network (CAN) - Part 1: Data link layer and physical signalling, 2021, https://www.iso.org/standard/63648.html. (Accessed August 2021).
[8] S. Chouali, A. Boukerche, A. Mostefaoui, M.A. Merzoug, Formal verification and performance analysis of a new data exchange protocol for connected vehicles, IEEE Trans. Veh. Technol. 69 (12) (2020) 15385–15397.
[9] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables devops: Migration to a cloud-native architecture, IEEE Softw. 33 (3) (2016) 42–52.
[10] J. Dixon, Pentaho, hadoop, and data lakes, 2021, https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. (Accessed August 2021).
[11] H. Fang, Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem, in: 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems, CYBER, IEEE, 2015, pp. 820–824.
[12] Apache Hadoop, 2021. https://hadoop.apache.org/. (Accessed August 2021).
[13] S. Acharya, B. Rajesh, S.B. Vaghani, I. Sokolinski, S. Chiao-Chuan, K. Desai, Object storage system, US Patent 8, 775, 773, 2014.
[14] Elasticsearch: Free and Open Search, 2021. https://www.elastic.co/. (Accessed August 2021).
[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: 9th {USENIX} Symposium on Networked Systems Design and Implementation, {NSDI} 12, 2012, pp. 15–28.
[16] J. Dean, S. Ghemawat, Mapreduce: A flexible data processing tool, Commun. ACM 53 (1) (2010) 72–77.
[17] R. Van Der Lans, Data Virtualization for Business Intelligence Systems: Revolutionizing Data Integration for Data Warehouses, Elsevier, 2012.
[18] J. Han, E. Haihong, G. Le, J. Du, Survey on nosql database, in: 2011 6th International Conference on Pervasive Computing and Applications, IEEE, 2011, pp. 363–366.
[19] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. 26 (2) (2008) 1–26.
[20] C. Strozzi, NoSQL: A non-SQL RDBMS, 2021, http://www.strozzi.it/. (Accessed August 2021).
[21] P. Adamczyk, P.H. Smith, R.E. Johnson, M. Hafiz, Rest and web services: In theory and in practice, in: REST: From Research to Practice, Springer, 2011, pp. 35–57.
[22] R.T. Fielding, Architectural Styles and the Design of Network-Based Software Architectures, University of California, Irvine, 2000.
[23] M.A. Serhani, H.T. El Kassabi, I. Taleb, A. Nujum, An hybrid approach to quality evaluation across big data value chain, in: 2016 IEEE International Congress on Big Data, BigData Congress, IEEE, 2016, pp. 418–425.
[24] Y.W. Lee, D.M. Strong, B.K. Kahn, R.Y. Wang, Aimq: A methodology for information quality assessment, Inf. Manage. 40 (2) (2002) 133–146.
[25] B.T. Hazen, C.A. Boone, J.D. Ezell, L.A. Jones-Farmer, Data quality for data science, predictive analytics, and big data in supply chain management: An introduction to the problem and suggestions for research and applications, Int. J. Prod. Econ. 154 (2014) 72–80.
[26] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soulé, et al., IBM streams processing language: Analyzing big data in motion, IBM J. Res. Dev. 57 (3/4) (2013) 7:1–7:11.
[27] A. Nassar, A. Mostefaoui, F. Dessables, Improving big-data automotive applications performance through adaptive resource allocation, in: 2019 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2019, pp. 1–7.
[28] O. Boykin, S. Ritchie, I. O'Connell, J. Lin, Summingbird: A framework for integrating batch and online mapreduce computations, Proc. VLDB Endow. 7 (13) (2014) 1441–1451.
[29] W. He, G. Yan, L. Da Xu, Developing vehicular data cloud services in the iot environment, IEEE Trans. Ind. Inf. 10 (2) (2014) 1587–1595.
[30] T. Häberle, L. Charissis, C. Fehling, J. Nahm, F. Leymann, The connected car in the cloud: A platform for prototyping telematics services, IEEE Softw. 32 (6) (2015) 11–17.
[31] A. De Iasio, A. Furno, L. Goglia, E. Zimeo, A microservices platform for monitoring and analysis of iot traffic data in smart cities, in: 2019 IEEE International Conference on Big Data, Big Data, IEEE, 2019, pp. 5223–5232.
[32] A. Krylovskiy, M. Jahn, E. Patti, Designing a smart city internet of things platform with microservice architecture, in: 2015 3rd International Conference on Future Internet of Things and Cloud, IEEE, 2015, pp. 25–30.
[33] A. d. M. Del Esposte, E.F. Santana, L. Kanashiro, F.M. Costa, K.R. Braghetto, N. Lago, F. Kon, Design and evaluation of a scalable smart city software platform with large-scale simulations, Future Gener. Comput. Syst. 93 (2019) 427–441.
[34] C. Badii, E.G. Belay, P. Bellini, M. Marazzini, M. Mesiti, P. Nesi, G. Pantaleo, M. Paolucci, S. Valtolina, M. Soderi, et al., Snap4City: A scalable IOT/IOE platform for developing smart city applications, in: 2018 IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI, IEEE, 2018, pp. 2109–2116.
[35] H. Khazaei, S. Zareian, R. Veleda, M. Litoiu, Sipresk: A big data analytic platform for smart transportation, in: Smart City 360°, Springer, 2016, pp. 419–430.
[36] L.F. Herrera-Quintero, J.C. Vega-Alfonso, K.B.A. Banse, E.C. Zambrano, Smart ITS sensor for the transportation planning based on IoT approaches using serverless and microservices architecture, IEEE Intell. Transp. Syst. Mag. 10 (2) (2018) 17–27.
[37] A. Javed, K. Heljanko, A. Buda, K. Främling, Cefiot: A fault-tolerant IoT architecture for edge and cloud, in: 2018 IEEE 4th World Forum on Internet of Things, WF-IoT, IEEE, 2018, pp. 813–818.
[38] J. Innerbichler, S. Gonul, V. Damjanovic-Behrendt, B. Mandler, F. Strohmeier, Nimble collaborative platform: Microservice architectural approach to federated IoT, in: 2017 Global Internet of Things Summit, GIoTS, IEEE, 2017, pp. 1–6.
[39] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, D. Ganguli, Druid: A real-time analytical data store, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014, pp. 157–168.
[40] F. Yang, G. Merlino, N. Ray, X. Léauté, H. Gupta, E. Tschetter, The radstack: Open source lambda architecture for interactive analytics.
[41] L. Sanchez, L. Muñoz, J.A. Galache, P. Sotres, J.R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, et al., Smartsantander: Iot experimentation over a smart city testbed, Comput. Netw. 61 (2014) 217–238.

[42] B. Cheng, S. Longo, F. Cirillo, M. Bauer, E. Kovacs, Building a big data platform for smart cities: Experience and lessons from santander, in: 2015 IEEE International Congress on Big Data, IEEE, 2015, pp. 592–599.
[43] S. Yamamoto, S. Matsumoto, S. Saiki, M. Nakamura, Using materialized view as a service of scallop4sc for smart city application services, in: Soft Computing in Big Data Processing, Springer, 2014, pp. 51–60.


Ahmed Mostefaoui is an associate professor at the University of Franche-Comte since 2000. Currently, he is also a visiting professor at Ottawa University (PARADISE Research Lab). Dr. Ahmed received the M.S. and Ph.D. degrees in computer science from Ecole Normale Supérieure de Lyon, respectively, in 1996 and 2000. In 2009, he obtained his HDR (Habilitation to Direct Research) from the University of Franche-Comte. His research interests are in distributed algorithms in wireless ad hoc and sensor networks and their security (emphasizing both practical and theoretical issues), multimedia systems and networking, in particular, large-scale distributed architectures.

MOHAMMED AMINE MERZOUG received the Ph.D. degree in computer science from the University of Bejaia, in 2019; thesis prepared jointly at the University of Bejaia and University Bourgogne-Franche-Comté. He is currently a Lecturer-Researcher with the University of Batna 2. He has been holding the position of associate professor at the Department of Computer Science, since May 2019. His current research interests include big data, cloud computing, distributed systems, distributed algorithms, machine learning, and WSNs/WMSNs.

Amir Haroun Holds a master's degree from the University of Rouen (France). He is actually a Senior Data architect at Stellantis Group, Poissy and a Ph.D. candidate at the university of Franche-Comte. He has been involved in the development of Big Data platform for connected cars at Stellantis Group.

Anthony Nassar received his Ph.D. from the university of Franche-Comté in 2020 and the Master's degree in Computer Science and Supply Chain from the Université de Pau et des Pays de l'Adour (UPPA), France. He is actually data architect at Stellantis Group. His research interests include Big data parallel computing, distributed systems and virtualization.

François Dessables is actually a Senior Data architect at Stellantis Group, Bessoncourt. He has been involved in the development of Big Data platform for connected cars at Stellantis Group.