A Big Data Architecture for Automotive Applications: PSA Group Deployment Experience

Amir HAROUN
RDD,
PSA Group
Bessoncourt, France
Email: amir.haroun@mpsa.com

Ahmed MOSTEFAOUI
FEMTO-ST Institute/CNRS
Bourgognes-Franche-Comte University
Montbéliard, France
Email: ahmed.mostefaoui@univ-fcomte.fr

François DESSABLES
DDCE,
PSA Group
Bessoncourt, France
Email: francois.dessables@mpsa.com

***Abstract*—Vehicles have become moving sensor platforms collecting huge volumes of data from their various embedded sensors. This data has a great value for automotive manufacturers and vehicles owners. Indeed, connected vehicles data can be used in a large broad of automotive services ranging from safety services to well-being services (e.g. fatigue detection). However, vehicle fleets send big volumes of data that traditional computing and storage approaches are not able to manage efficiently. In this paper, we present the experience of the PSA Group[1] on leveraging big data in automotive context. We describe in depth the big data architecture deployed within the PSA Group and the underlaying technologies/products used in each component.**

***Keywords*-Big Data; Reference Architecture; Connected Vehicles;**

I. Introduction

Recent years have witnessed a great advance in connected mobile devices including connected vehicles. Connected Vehicles (CVs) are able to remotely connect and communicate with their surroundings. This connectivity is not limited to other vehicles (Vehicle-To-Vehicle communication (V2V)), CVs can also communicate with an infrastructure (Vehicle-To-Infrastructure (V2I) and Infrastructure-To-Vehicle (I2V)) or with other devices (V2X) such as mobile phones. CVs send a large range of data collected from the various sensors deployed on its components. Today, PSA Group vehicles are sending more than 170 information (e.g., the Vehicle Identification Number (VIN), Rounds Per Minutes (RPM), GPS coordinates, etc). This wide range of data can be of a great value to automotive manufacturers and vehicles owners. Indeed, leveraging CV's data can improve the driver's safety, vehicle services and the mobility experience.

However, CV fleets send huge volumes of data[2] that are difficult to handle using traditional approaches [**?**]. Processing and analyzing this huge volumes of data is considered as a *big data* challenge. Big data is a term used to describe huge data sets that are difficult to handle efficiently using traditional processing and storage approaches due to their complexity (refer to Section II for detailed description).

The automotive ecosystem is based on sharing data to obtain remote services. In addition to the valuable benefit for automotive manufacturers such as predictive and preventive analysis, CVs data can be used in large range of customer services such as [**?**] safety, mobility management, vehicle management, entertainment, autonomous driving, etc. All the generated data, which can be of an enormous volume, is usually send to the infrastructure in order to be processed and stored for future analysis.

All these intrinsic data characteristics have motivated the design and the development of novel frameworks that are able to process, store and exchange data with clients including third party applications, fleet managers and partners as shown in Figure 1. We note an additional important issue that has to be taken into account when managing vehicles data is the driver's privacy protection by enabling and setting security and privacy policies to access and share this sensitive data.

Hence, the expected framework has to consider the following requirements:

- **Scalability:** The proposed architecture must be highly scalable and maintain a good performance to deal with the increasing number of connected vehicles.
- **Robustness:** The proposed architecture must be robust and fault tolerant, if any thing happens in any level, service recovery must be immediate, fully transparent, and without any data loss.
- **Real-time processing:** Must be able to support data processing in both online and batch mode to cover all the services that require real-time processing or historical offline processing. The *online* processing processes fresh data as it arrives from vehicles in real-time or near real-time. The *off-line* or *batch* processing is dedicated to compute historical data with high latency.
- **Data sharing:** To exploit CV's data full potential, commercial partnerships should be considered for deployment. Vehicle owners must be able to grant access and share their data to partners or third party applica-

[1]PSA Group is the second-largest automobile manufacturer in Europe with about 3 million sold vehicles in 2015.

[2]By 2020, the estimated number of PSA Group's connected cars is about 5 millions.

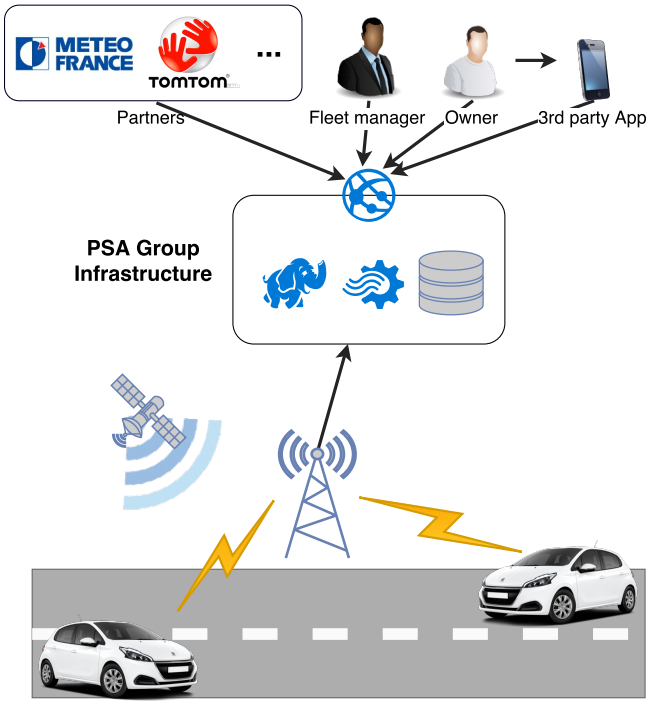

Figure 1. PSA Group services

tions and control what information can be accessed for what partner.

Although many initiatives have been conducted by researchers on leveraging CVs data [**?**], [**?**], to the best of our knowledge there is no big data architecture design for automotive applications that meets all the requirements. The idea of this paper came from the fact that this is a recent domain and there is feedback or guideline for automotive manufacturers on the topic. The contribution of this paper is to try standardizing an architecture design by providing PSA Group's feedback on leveraging CVs data. To this end, architecture in automotive context, and general purpose big data architectures are reviewed. Subsequently, a detailed description of the PSA Group's architecture and its underlaying technologies/products is done.

The rest of the paper is as follows: Section II provides an overview of big data technologies, Section III presents the related work, a detailed description of our architecture and its core components can be found in Section IV. Section V presents two automotive services deployed on PSA Group's platform. Finally, we conclude this paper and highlight the future work in Section VI.

II. Big Data Technologies Overview

According to Gratner [**?**] "Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation". Big Data is referred to large and complex datasets that requires new processing techniques due to their characteristics as well known as 3V (i.e. volume, velocity, and variety). However, to define more precisely the Big Data scope, two more '*Vs*' are added:

- Volume: Stands for the growing size of datasets. In fact, 90% of the data is generated in the two past years. This issue makes the traditional processing and storage technologies obsolete in the face of this huge amount of data.
- Velocity: The problem of velocity is that data is generated at a high frequency that makes the processing and storage solution slower than the generation rate.
- Variety: Data is collected from a variety of heterogeneous sources with different formats. It can be structured, semi-structured, or non-structured (sound, image, video, etc).
- Veracity: Addresses the trustworthiness of data. Data can be uncertain, incomplete, or inconsistent.
- Value: The value is manifested through the different usages that can be made (e.g. predictive analysis, CRM, etc).

Big data challenges include: capture, storage, analysis, sharing, and visualization.

To overcome these Big Data characteristics, new technologies have emerged. We categorize them in 3 domains: NoSQL, storage, and processing.

A. NoSQL

The term NoSQL was first used by Carlo Strozzi in 1998 [**?**] and defined as a derivative of the relational database system. According to C. Strozzi, NoSQL uses shell-level tools and stores data in regular UNIX ASCII files. The late NoSQL defines a class of Database Management Systems (DBMS) designed for Big Data. There is no precise definition of NoSQL, however NoSQL databases have three common characteristics: high scalability, data replication, and schema-less data model. We explain these three features bellow.

- Scalability: is achieved by distributing data over clusters of machines to deal with the increasing size of data.
- Replication: data is replicated on the different machines of a cluster in order to achieve redundancy.
- Schema-less: Unlike traditional RDBMS that requires a predefined table schema, in NoSQL databases there is no schema.

To handle the different characteristics of big data, there are four families of NoSQL databases [**?**]: Key-Value, Column-Oriented, Document-Oriented, and Graph databases.

B. Storage

The most compelling DFS in big data is the Hadoop implementation of the Google File System namely Hadoop Distributed File System (HDFS). HDFS is a distributed file system inspired from the Google File System (GFS) that runs on commodity hardware. HDFS has a master/slave architecture, an HDFS cluster consists of one NameNode (master node that manages the cluster and stores the files meta-data) and multiple DataNodes (slaves nodes that store data blocks). A file is split into blocks then these blocks are spread on the different DataNodes for storage. Although its scalability and high availability features, HDFS has some industrial limits.

- HDFS is not POSIX-compliant. In order to use the data stored within HDFS, you must use the HDFS shell or Hadoop APIs, this makes the daily operations very difficult:
  - HDFS does not support the current working directory and hence the full path must be specified at each operation.
  - To compare two files, you have to copy them from HDFS to local then use the UNIX *diff* command:

```
diff <(hdfs dfs -cat /path/to/file1) <(hdfs dfs
    -cat /path/to/file2)
```

- HDFS is designed for large files rather than small files.
- Data stored on HDFS is immutable (read-only). To modify it, it is necessary to move the file out of HDFS, modify it and then move it back into HDFS.
- The NameNode machine is a single point of failure (SPOF) for an HDFS cluster. If the NameNode machine fails, the service will be lost and manual intervention is necessary [**?**].

General Parallel File System (GPFS) [**?**] is a fully POSIX, kernel level Distributed File System (DFS). It has no single point of failure, it overcomes the HDFS NameNode failure by distributing both data and meta-data across the disks of cluster also called Network Shared Disks (NSDs). GPFS maintain the configuration information in a dedicated node called GPFS cluster configuration server, the failure of the cluster configuration server results in the failure of GPFS administration commands.

C. Parallel processing

The late big data high-level processing frameworks democratized the parallel processing by providing a high level abstraction of resources of the cluster. Based on latency requirements, two types of processing can be distinguished: high latency processing and low latency processing.

1) Batch processing: Batch processing is the general form of processing. It's designed for complex treatments on historical data such as analytics. It is generally more concerned with throughput than latency of individual components of the computation.

Map Reduce [**?**] is one of the most adopted frameworks in the field of batch processing, it's composed of two separate but dependent set of tasks that a MapReduce job will perform (refer to Figure 2). The first tasks are the mappers, they split the data into chunks and process them in parallel to generate key-value pairs (tuples). The second set of tasks are reducers, they take the output of the map tasks as input and combine this pairs to get a smaller set of tuples. Whereas traditional parallelism approaches bring data to the computation, MapReduce does completely the opposite by bringing computation to the data "*Moving computation is cheaper than moving data*" [**?**].
MapReduce is an efficient solution for *one-pass computation* but when it comes to multi-pass computation (i.e., executing multiple Map Reduce jobs in sequence), Map Reduce is inefficient due to the high latency of disk operations (i.e. read and write).

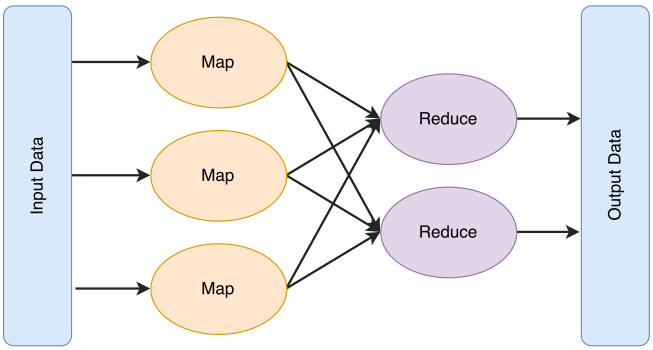

Figure 2. A MapReduce Job

While MapReduce is not designed for multi-pass computation (e.g., iterative machine learning), Spark is defined as streaming's batch processing (even though Spark has a streaming library). Spark claims to be 100x faster than Map Reduce in memory and 10x faster on disk [**?**]. Moreover, Spark offers flexibility of development's languages: Scala, Java, Python, R.
Spark implements Resilient Distributed Dataset (RDD) [**?**], to provide a distributed memory abstraction. RDD represents an immutable, partitioned collection of elements that can be operated on in parallel. Data can be loaded also as Data Frames (DF), they have a similar conception to a table in relational databases. Unlike RDD, DF provides a rich information about the stored data, Spark uses this extra-information to improve performance.

2) Stream processing: In contrast to batch processing, stream computing is a new programming paradigm designed for distributed and parallel processing of unbounded data streams (not to confuse with real-time processing that requires a response within a certain period of time).
To understand better the stream processing, we describe some of its main characteristics:

- Each and every data item is processed as it arrives (i.e. online).
- Events are time-based, every record is typically timestamped on creation.
- Operations are done in a data flow design.
- Every operation is done on one data element (or a small window of recent data).
- An operation computes something relatively simple.
- Each computation needs to complete in (near) real-time to avoid congestions.

The hard constraint of stream computing is that all the operators must have a processing rate bellow (or at least equal to) the input rate to guarantee that the input data will be processed as it comes. For instace, if the data's generation frequency is 100.000 event/s and there is just one operator that processed 90.000 event/s, this will cause congestion that can go up to the source.

To achieve these results, stream computing exposes naturally various levels of parallelism (i.e. pipeline, task, and data parallelism). Streaming applications are programs that process continuous data streams, they are composed of a set of operators instances interconnected with stream connections. A streaming application can me modeled as a directed graph where vertices are data streams (i.e. continuous series of tuples, generated by an operator) and edges are operators instances (see Figure 3). There are three types of operators:

- *Source operators:* sources of the input data streams. They are connected to an external source of data so they transform the external data into data streams to send them via their output ports.
- *Transformation operators:* carrying out the computation, they transforms input data streams to output data streams by applying some operations (e.g., aggregation, correlation, etc).
- *Sink operators:* having only input ports, they are usually placed at the end of the streaming application.

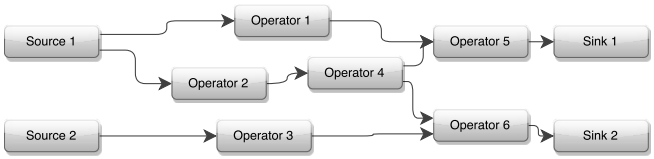

Figure 3. A streaming application

Streaming applications are executed as *jobs* on Stream Runtime Environment (SRE) known also as instance. An instance can be deployed on a single processing node or multiple processing nodes, see Figure 4. A job is a running application on a SRE, it executes itself a set Processing Elements (PEs) executing each a set of operator instances. Given that jobs can be executed on multiple hosts, PEs within a job communicate using Inter-Process Communication (IPC). Samza [**?**] uses apache Kafka as IPC, IBM Streams [**?**] uses TCP as a IPC, etc.

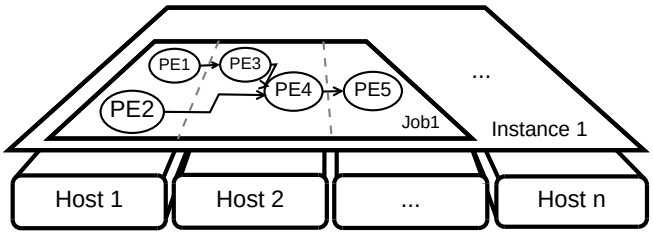

Figure 4. A Stream runtime environment (instance)

III. Related work

One of the earliest papers that introduced the topic of data sharing in telematics is presented in [**?**]. S. Duri et al. have defined some telematic services and policies to share and access CVs data. They have proposed a flexible privacy policy engine that allows data owners to specify access policies to their personal data, this minimize the interaction with the driver. Their privacy model does not only determines who can access data, but also for what purpose, under what constraints, for how long the data can be kept, and even to whom this data can be distributed. This framework gives a good idea of the constraints and use cases of sharing personal data it's however not used in our architecture due to its complexity and poor performance.

M. Johanson et al. presented the Big Automotive Data framework (BAuD) [**?**]. It is a framework designed for capture and online analysis of CV's data that covers some automotive needs (i.e. extensive CAN bus monitoring, remote diagnostic read-out, vehicles state of health).
The core components of the BAuD framework are:

- A telematic service platform,
- A cloud-based back-end infrastructure,
- A Task Manager,
- A Data Broker,
- An analytics service architecture,
- A web-based user interface front-end.

Data is sent from the telematic service platform as Measurement Data Format (MDF), then it's stored on a distributed file system. This data is then analyzed using the analytic framework based on an analytics task definition. A web based user interface is used to interact with the framework. Although this framework meets well to some automotive needs, it does cover all the automotive requirements such as real-time analysis, data sharing, etc.

Another general purpose big data architecture is the Lambda Architecture (LA) [**?**] proposed by N. Marz. LA is a generic scalable big data architecture design that handles massive volumes of real-time and historical data. It computes similar analytics twice: the high-latency historical processing and low-latency processing on recent data. Hence, LA is composed of three main layers: speed layer, batch layer and

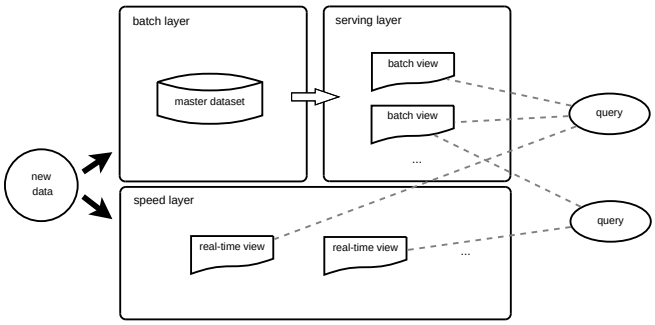

Figure 5. The Lambda Architecture [**?**]

serving layer (see Figure 5). The new data is dispatched to the two processing layers (i.e. batch and speed layer). The batch layer has two main functions: storing the master data-set as immutable, append-only set of raw data and pre-computing the batch views. The serving layer indexes the data stored on the batch layer so it can be queried. The speed layer compensates the high latency of the batch layer by computing low latency speed view. Merging results from the batch and the speed views allows answering any query. The LA takes advantages from the precise historical processing in the batch layer without losing the low latency information provided by the speed layer. However, its very difficult to keep the processing treatments perfectly synchronized on two complex and different frameworks (i.e speed and batch) [**?**], any modification in one layer must be reported in the other layer otherwise it causes problems in merging the two results. Indeed, speed and batch layers adopt completely different approaches of processing: the batch layer processes the data in rest and the speed layer processes the data in motion.

IV. PSA Group's big data architecture

In this section, we describe our big data reference architecture for automotive application, its main components and the technologies/products that can be used in each part. Inspired from the LA, we propose an architecture based on its three layers (i.e., speed, batch, and serving layer). We have however separated the high latency applications in the batch layer from those requiring low latency in the speed layer, this removes the need of maintaining the speed layer and the batch layer synchronized.

The first step to leverage CV's data is to capture and send this data, this is done using Telematic Service Units (TSUs). The key role of TSUs is to capture data streams from the vehicle and send them to the infrastructure via wireless mobile data communication. TSUs are installed on the vehicles and connected generally to the Controller Area Network (CAN) [**?**] bus [3].

[3]The ISO 11898 standard specifies that the CAN physical layer allows transmission rates up to 1 Mbit/s for use within road vehicles. See http://www.iso.org/iso/catalogue_detail.htm?csnumber=63648

PSA Group uses several types of TSUs (e.g. Autonomous Telematic Boxes (ATB), approved aftermarket boxes, etc). Each of these units has special applications, some units are designed for extensive monitoring so they send a large range of data at a very high frequency while other units are designed for fleet management, etc. There are some after-market TSUs that provide other features such as GPS location for vehicles that are not equipped with this feature, so they enhance the CV's data with additional information.

In addition to the TSUs, we can use the V2X communication (using Bluetooth) to send the vehicle's data to a smart-phone then to the infrastructure. Data is sent in binary coded format using the various communication protocols (e.g.,HTTP, MQTT, AMQP, etc).

In order to simplify our architecture, we present the components into functional layers.

- Device and service management layer.
- Frontend layer that receives the data and decode it,
- Message Queue (MQ) layer that provides a resilient data feed for the rest of the components,
- Speed layer that processes the data in (near) real-time,
- Batch layer that stores and processes the data in batch mode,
- Serving layer, to index and expose the data for querying.

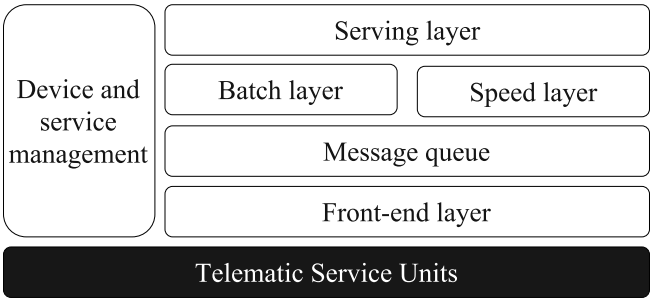

Figure 6. PSA Group's reference architecture

A. Device and service management layer

We centralized all management functions (e.g., devices, partners, services, etc) into a single component to act as a reference information. This layer consists of a database containing all information and web interfaces to interact with this database. Vehicle owners, administrators, and partners have access into their interface. This information is crucial for automotive applications. In fact, the type of processing and transformation applied on CVs data is based on this reference information.

B. Frontend layer

Depending on the communication protocol used by TSUs, data have to be ingested and decoded before it can be used, this is done in the frontend layer.

For instance, TSUs using HTTP/HTTPS as communication protocol, the frontend layer is an HTTP web server that hosts web applications. The web applications receive the HTTP incoming flow, decode it, and transform it into MQ messages to finally transfer it to the MQ layer.

C. MQ layer

To provide a resilient data feed for processing layers (speed layer and batch layer), we use a MQ layer. MQ is a message-oriented middleware that acts as a producer-consumer queue to provide an asynchronous and secured communication. The sender and the receiver does not interact directly, meanwhile messages are kept in memory or stored on disc (for persistence) until the receiver retrieves them. This ensures a resilient data feed that covers the temporary unavailability of the sender or the receiver.

D. Speed layer

The speed layer is the most important component in the architecture, it is responsible of the online processing, the continuous data flows with hard constraints (i.e., real-time or near real-time computing). In order to fulfill these constraints, we use the stream computing paradigm (described in Section II). This layer is also responsible of preparing the raw data for storage on the batch layer for further analysis. Due to the fact that CV's data is personnal, the French control authority for the protection of personal data (CNIL) imposes laws for the protection of this private data.In fact, CV's sensitive data (e.g., VIN, location, speed, etc) can't be stored as it arrives. Hence, data is anonymized, filtered, aggregated, corresponding to customers preferences. To do so, every customer has a contract in which all his preferences are listed such as anonymization, partners that can access to his data, the shared data, etc. This contract is accessed via web and can be modified any time, these modification must be taken into account in the speed layer. For instance, the speed information cannot be shared as it comes, and an aggregation on a group of vehicles or an average on a window of time must be done before sharing it. The stream computing's functional architecture consists of micro-services [**?**], where each micro-service is a streaming application that has a particular function and communicates with other micro-services (see Figure 7).
The main micro-services are:

- Acquisition: connected to the MQ layer and retrieves fresh data as it arrives.
- Dispatcher: receives CVs data (from acquisitions) and referential data (from the device and service management layer), role preparation, anonymization, matching with contracts, etc
- Services: micro-services that are provided to vehicle owners and third-party partners.
- Storage: all the previous micro-services have access to the storage micro-service, it is responsible of storing data on the DFS.

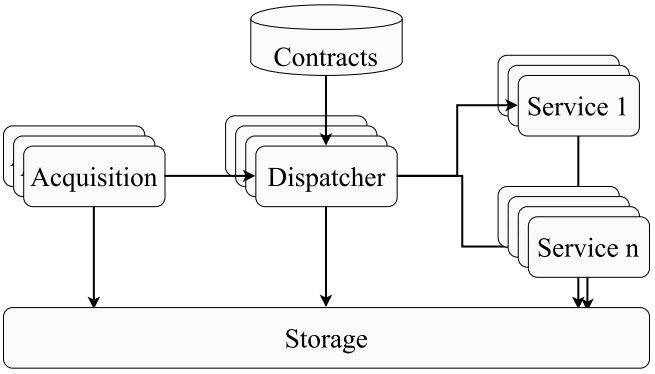

Figure 7. The functional architecture of speed layer

E. Batch layer

The batch layer of our architecture has two main functions: batch processing and storage). This layers does not have however the hard real time processing constraints of the speed layer, hence it adopts a completely different framework. Typically, the batch layer can be implemented using the open-source Hadoop framework [**?**]. Inspired from the Google's MapReduce [**?**] and Google File System [**?**], Hadoop offers similar components (i.e. Hadoop MapReduce and Hadoop Distributed File System (HDFS) for processing and storage respectively).

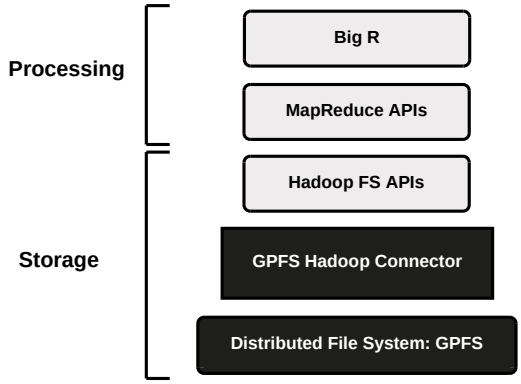

Figure 8. The batch layer

1) Storage: In the original LA, the storage component is responsible of storing the master dataset as immutable, append-only set of raw data. In automotive context, there are however privacy constraints that prevent storing raw data. Hence, data has to first *pre-processed*[4] before be stored as immutable, append only data. To store the huge volumes of

[4]Pre-processing is done in the speed layer and includes anonymization, filtering, aggregation, etc.

data generated by CV fleets, a DFS must be used. The DFS is responsible of managing large volumes of data effectively in large clusters. It ensures scalability by its distributed nature (i.e. it scales by adding more machines to the cluster), it ensures also high availability through data stripping and replication among the different nodes of the cluster.
To overcome the HDFS limitations, we used the DFS GPFS. Since GPFS is a non Hadoop standard component, another layer called GPFS Hadoop connector must be add on top of GPFS (see Figure 8). It acts as an adaptor between Hadoop's other components and GPFS, so it allows Hadoop to access data from a GPFS file system as if it is on HDFS.

2) Processing: While the speed layer processes the new data as it arrives, this layer is responsible of processing historical data with high-latency jobs. In our deployment we used the Map Reduce as a distributed programming model in combination with BigR. BigR uses the classic R language to perform data analysis on big data without writing complex Map Reduce jobs. For instance, the Eco-driving service (refer to Section V) consists of a daily job that generates a classification model from the historical data.

There are several enterprise distributions based on the Hadoop framework (e.g. IBM BigInsights, MapR, Cloudera, HortonWorks, etc) that propose other features such as administration tools, deployment facilities, etc. In our framework, the batch layer is implemented using the IBM BigInsights distribution of Hadoop.

F. Serving layer

To leverage CVs data, processed data from the batch layer and the speed layer must be indexed and served to external users. In this layer, we define access policies, who (Business-To-Business (B2B), Business-To-Customer (B2C), etc) can access to what resource (geographical data, environmental data, etc). We used two different types of NoSQL databases [**?**] to index the processed data, Apache HBase for a real-time random read-write and BigSQL for a batch queries.
Apache HBase [**?**] is a distributed, versioned, column oriented NoSQL database modeled after the Google's BigTable [**?**]. The HBase database is used for storing and querying the (near) real-time views of the speed layer.
BigSQL [**?**] is a SQL engine to access data stored on Hadoop. BigSQL provides a full ANSI SQL: this means that data is queried via the standard SQL language regardless of where it is stored on Hadoop (e.g. HDFS, HBase, etc). It has the capability to use the MapReduce parallelism for the big ad-hoc queries. A single query can execute multiple MapReduce jobs.

We Exposed CVs data on the web using Application Programming Interfaces (APIs) [**?**] as RESTful [**?**](i.e., systems conform to the constraints of REST) web services on top of the NoSQL databases (refer to Figure 9). APIs provide a controlled access to a specific set of data (referred as resource). Each resource is identified by a unique Uniform

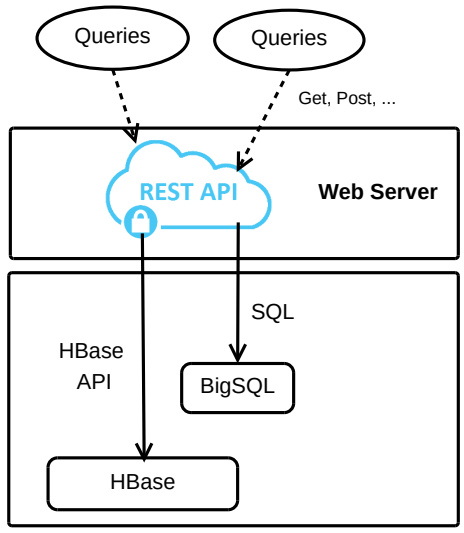

Figure 9. The Serving Layer

Resource Identifier (URI); e.g. Maintenance, Ecodriving, Crash, Safety, Environment, etc.

In our deployment, third-party clients (e.g. mobile applications) can also access CVs data. To grant referenced third-party clients access to personal CV data without exposing the user's credentials, we used the Open Authorization (OAuth) protocol. OAuth provides a delegated access to a resource on behalf of the resource owner, this is done by providing access tokens to third-party clients with the approval of the resource owner. The token allows third-party clients to access protected resources on the resource server. A simple example of the flow is presented in Figure 10, for the sake of simplification the authentication and resource servers are represented as a single component.

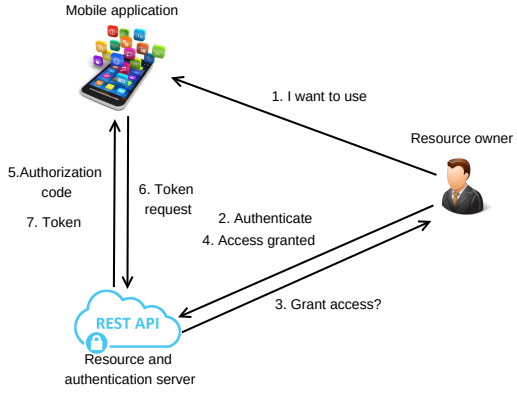

Figure 10. The Open Authentication

To manage APIs, there are dedicated frameworks called API management tools. They provide a full life cycle governance of APIs, from creation, publication, securing and monitoring.

In our implementation, we used IBM API Connect [5] for API management and Swagger specifications [6] for API creation. An Open Source solution is WSO2 API Manager [7].

V. Deployment examples

In this section, we will describe two services in production deployed on PSA Group's platform.

A. Temperature

The temperature service [**?**] is requested by the French National Meteorological Service [**?**] for their meteorological previsions, it aims to provide a real time overview of the whole country temperature. The objective is to get an updated map of the country temperature, including regions and departments, as precise as possible (see Figure 11). Hence, each CV sends the captured exterior temperature, tagged with its current location to the infrastructure. The latter is then responsible of correlating the received data in order to continuously updating the map.

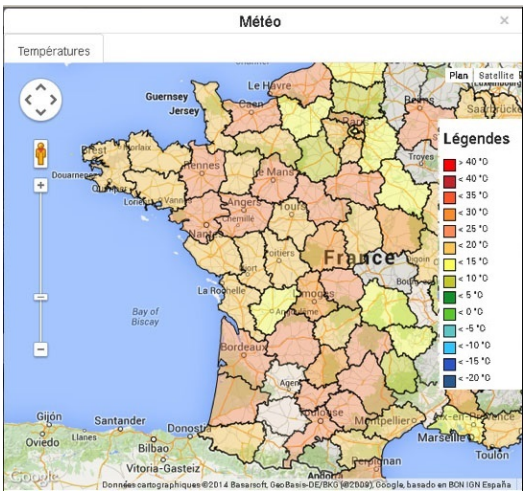

Figure 11. The temperature service

B. Eco-coaching

This service analyses the driving style and correlates it with the environment and other people's drivings in order give a note and advises to improve the driving. This service is composed of two operations: a batch job that generates a model based on the historical data and a streaming application that uses the model (generated by the batch job) to evaluate the driving style (refer to Figure 12). This combination of a batch job and a streaming job grantees an accurate evaluation thanks to the historical processing and an online result with the streaming application. This service does not only reduce the CO2 emissions and fuel consumption [**?**], it can also extend the life cycle of the vehicle parts by having an optimized usage of the vehicle.

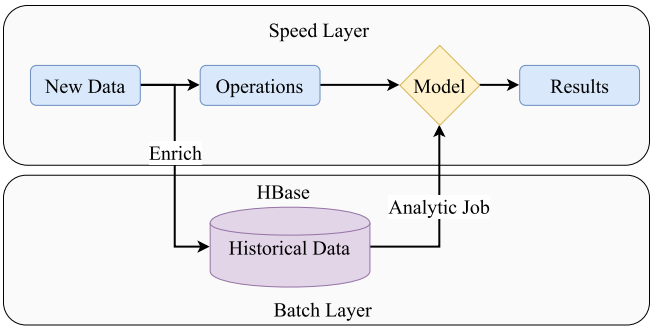

Figure 12. The eco-coaching service

VI. Conclusion

This paper presents an overview of big data architecture deployed within PSA Group for automotive applications. This architecture has primarily been designed to respond to specific automotive applications requirements. First, a description of applications and challenges of leveraging CVs data has been done. Then all parts and components are described by highlighting the Subsequently, PSA Group's big data architecture is deeply reviewed by technologies/products chosen in each part. However, this architecture is unidirectional, and for some needs such as vehicle management, we want also to send informations to the vehicle itself. Currently, we are working on a car to car architecture ovec the cloud [**?**] using MQTT [**?**]. Typically, CVs will publish their data and subscribe to topics, this will simplify the interaction between the infrastructure and the vehicles. Another improvement would be use Apache Spark for processing and analytics to improve the performance of batch processing.

[5] http://www-03.ibm.com/software/products/en/api-connect
[6] http://swagger.io/
[7] http://wso2.com/api-management/