Optimizing resource utilization in distributed computing systems for automotive applications

Anthony Nassar

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

UNIVERSITÉ DE FRANCHE-COMTÉ

THÈSE DE DOCTORAT

DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ

École doctorale n°37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

par

ANTHONY NASSAR

Optimizing Resource Utilization in Distributed Computing Systems for Automotive Applications

Optimisation de l'utilisation des ressources dans les systèmes informatiques distribués pour les applications automobiles

Thèse présentée et soutenue publiquement le 04-02-2021 à Belfort,
devant le Jury composé de :

MR CERIN CHRISTOPHE	Professeur à l'Université Sorbonne Paris Nord	Président
MR CHBEIR RICHARD	Professeur à l'Université de Pau et des Pays de l'Adour	Rapporteur
MME BENBERNOU SALIMA	Professeur à l'Université Paris-Descartes	Rapporteur
MR MOSTEFAOUI AHMED	Maître de conférences à l'Université de Franche-Comté	Directeur de thèse
MR DESSABLES FRANÇOIS	Ingénieur chez Groupe PSA	Codirecteur de thèse

DOCTORAL THESIS

OF THE UNIVERSITY BOURGOGNE FRANCHE-COMTÉ INSTITUTION

PREPARED AT UNIVERSITÉ DE FRANCHE-COMTÉ

Doctoral school n°37

Engineering Sciences and Microtechnologies

Computer Science Doctorate

by

ANTHONY NASSAR

Optimizing Resource Utilization in Distributed Computing Systems for Automotive Applications

Optimisation de l'utilisation des ressources dans les systèmes informatiques distribués pour les applications automobiles

Thesis presented and publicly defended in Belfort, on 04-02-2021

Composition of the Jury :

CERIN CHRISTOPHE	Professor at Université Sorbonne Paris Nord	President
CHBEIR RICHARD	Professor at Université de Pau et des Pays de l'Adour	Reviewer
BENBERNOU SALIMA	Professor at Université Paris-Descartes	Reviewer
MOSTEFAOUI AHMED	Associate Professor at Université de Franche-Comté	Supervisor
DESSABLES FRANÇOIS	Engineer at Groupe PSA	Co-supervisor

ABSTRACT

Optimizing Resource Utilization in Distributed Computing Systems for Automotive Applications

Anthony NASSAR
University of Bourgogne-Franche-Comté, 2020

Supervisors: Ahmed Mostefaoui, François Dessables

One of the main challenges for the automobile industry in the digital age is to provide their customers with a reliable and ubiquitous level of connected service, which is increasingly expected and has become the norm in recent years. As we have today's smartphones and smart fridges, the automotive world would not be an exception. Smart cars have been entering the market for a few years now to offer drivers and passengers safer, more comfortable journeys, and especially more entertaining. All this by designing, behind the scenes, high-performance IT systems while economizing these resources. This CIFRE thesis, conducted in collaboration with Groupe PSA, a French automobile manufacturer that includes the automobile brands Citroën, DS Automobiles, Peugeot, as well as Opel and Vauxhall, aims to improve the allocation of platform resources through new sources of relevant information.

The Connected Vehicle concept, which appeared in the 2000s, and consisted of equipping vehicles with a layer of intelligence by connecting them to digital knowledge networks, the Internet, gave automakers the ability to offer new services to drivers.

The performance of a Big Data architecture in the automotive industry relies on keeping up with the growing trend of connected vehicles and maintaining a high quality of service. The PSA Cloud has a particular load when it comes to providing a real-time data processing service for all the brand's connected vehicles. With around 200k connected vehicles sold each year, the infrastructure is continuously challenged. Therefore, our two contributions aim to optimize the allocation of resources taking into account the specifics of continuous flow processing applications and to propose a modular and fine-tuned component architecture for automotive scenarios.

First, we go over a basic and essential process in Stream Processing Engines, a resource allocation algorithm. The central challenge of deploying streaming applications is how to map the operator graph, representing the application, to the available physical resources to improve the performance of the application (improved throughput, reduced processing time). We have targeted this problem by showing that the approach based on inherent data parallelism does not necessarily lead to all applications' best performance. In fact, through a real-world application (i.e., the Eco-Driving service), we have shown that a mapping based on insightful analysis of the specifics of the target application and the infrastructure's functionality can dramatically improve application performance, thus leading to added value. Our real experience has shown that our work improves the throughput compared to the straight-forward approach by around 4%. This improvement enables PSA's infrastructure to manage nearly 800,000 additional vehicles out of the 20 million Connected Vehicles expected by 2025.

Second, we revisit the Big Data architecture and design an end-to-end architecture that meets today's data-intensive applications' demands. Today, connected vehicles (CVs) can collect up to 170 different information (such as speed, temperature, fuel consumption) from integrated onboard sensors and transmit them, in real-time, to an infrastructure, usually by 4G/5G wireless communications. This reality raises many opportunities for developing new and innovative telematics services, including, among others, driver safety, customer experience, quality and reliability, location-based services, dealer services, infotainment, etc. It is expected that there will be around 2 billion connected cars by the end of 2025 on the world's roads, each of which can produce up to 30 terabytes of data per day. In real-time or batch mode, the management of this Big Data imposes strict constraints on the underlying data management platform. In this work, we report on VC's real Big Data platform, particularly the one deployed by Groupe PSA. In particular, we present open source technologies and products used in different components of the platform to collect, store, process, and, most importantly, exploit big data and highlight why the Hadoop system is no longer the *de-facto* solution of Big Data.

This thesis contributes to the improvement of resource allocation techniques on cloud platforms through workload characterization, the study of the specificity of deployed applications, and the evaluation of processing operator placement algorithms. Secondly, it details an end-to-end Big Data architecture ideal for automotive industry workflows. The studied algorithm cannot adapt to the dynamically variable data flow. Nevertheless, we believe that such a level of adaptability could be useful, especially to preserve a high level of service with good data flow and less downtime due to multiple compilations.

KEYWORDS: Big Data Architecture, Query Optimization, Cloud Computing, Hadoop Ecosystem, Stream Computing, Connected Vehicles

RÉSUMÉ

Optimisation de l'utilisation des ressources dans les systèmes informatiques distribués pour les applications automobiles

Anthony NASSAR
Université de Bourgogne-Franche-Comté, 2020

Superviseurs: Ahmed Mostefaoui, François Dessables

L'un des principaux défis de l'industrie automobile à l'ère du numérique est de fournir à ses clients un niveau de service connecté fiable et omniprésent, qui est de plus en plus attendu et qui est devenu la norme ces dernières années. La plupart de nos équipements aujourd'hui se transforme en des smart objets, ex., des smartphones, des smart-frigos, et le monde automobile ne serait pas une exception et les smart cars depuis quelques années font leur entrée sur le marché pour proposer aux conducteurs et aux passagers des trajets plus sûrs, plus confortables et particulièrement plus amusant. Tout cela en concevant, dans les coulisses, des systèmes informatiques performante tout en économisant l'utilisation de ses ressources. Cette thèse CIFRE, menée en collaboration avec Groupe PSA, un constructeur automobile français qui comprend les marques automobiles Citroën, DS Automobiles, Peugeot, ainsi que Opel et Vauxhall, vise à améliorer l'allocation des ressources de la plateforme grâce à de nouvelles sources d'informations pertinentes.

Le concept de Véhicule Connecté, apparu dans les années 2000, et qui consistait à donner une couche d'intelligence aux véhicules en les branchant aux réseaux de connaissance digitales, l'Internet. En faisant cela, les constructeurs automobiles proposent de nouveaux services aux conducteurs.

La performance d'une architecture Big Data dans l'automobile repose sur la capacité de suivre la tendance croissante des véhicules connectés et maintenir une qualité de service élevé. Le Cloud PSA possède une charge particulière lorsqu'il s'agit d'assurer un service temps réel de traitement de données pour tous les véhicules connectés de la marque : avec 200k véhicules connectés vendus chaque année, l'infrastructure est constamment mise au défi. C'est pourquoi nos deux contributions visent à optimiser l'allocation

des ressources en tenant compte des spécificités des applications de traitement de flux continu et de proposer une architecture de composants modulaires et peaufiné pour les scénarios automobiles.

Premièrement, nous passons en revue un processus de base et très essentiel dans les Stream Processing Engines, qui est un algorithme d'allocation de ressources. Le défi central du déploiement d'applications de streaming est la manière de mapper le graphe des opérateurs, représentant l'application, aux ressources physiques disponibles afin d'améliorer les performances de l'application (amélioration du débit, réduction du temps de traitement). Nous avons ciblé ce problème en montrant que l'approche basée sur le parallélisme des données inhérent ne conduit pas nécessairement aux meilleures performances pour toutes les applications. En fait, à travers une application du monde réel (i.e., le service Eco-Driving), nous avons montré qu'une cartographie basée sur une analyse perspicace des spécificités de l'application cible et des fonctionnalités de l'infrastructure peut améliorer considérablement les performances de l'application, conduisant ainsi à une valeur ajoutée. Nos expériences réelles ont montré que notre proposition s'améliore par rapport à l'approche directe d'environ 4% en termes de débit. Cette amélioration permet à l'infrastructure de PSA de gérer près de 800000 véhicules supplémentaires sur les 20 millions de Véhicules Connectés attendus d'ici 2025.

Deuxièmement, nous revisitons l'architecture Big Data et concevons une architecture de bout en bout qui répond aux exigences actuelles des applications gourmandes en données. Aujourd'hui, les véhicules connectés (VC) sont capables de collecter jusqu'à 170 informations différentes (vitesse, température, consommation de carburant, etc.) à partir de capteurs intégrés embarqués et de les transmettre, en temps réel, à une infrastructure, généralement par communications sans fil 4G/5G. Cette réalité soulève de nombreuses opportunités pour développer des services télématiques nouveaux et innovants, y compris, entre autres, la sécurité des conducteurs, l'expérience client, la qualité et la fiabilité, les services géolocalisés, les services des concessionnaires, l'infodivertissement, etc. On s'attend à ce qu'il y en ait environ 2 milliards voitures connectées d'ici fin 2025 sur les routes du monde, dont chacune peut produire jusqu'à 30 téraoctets de données par jour. La gestion de ce Big Data, en mode réel ou par lots, impose des contraintes strictes à la plateforme de gestion de données sous-jacente. Dans ce travail, nous faisons rapport de la plateforme de Big Data de VC, en particulier celle déployée par le Groupe PSA. En particulier, nous présentons des technologies et des produits open source utilisés dans différents composants de la plate-forme pour collecter, stocker, traiter et, plus important encore, exploiter le Big Data et souligner pourquoi le système Hadoop n'est plus la solution de facto Big Data.

En résumé, cette thèse contribue, en premier lieu, à l'amélioration des techniques d'allocation de ressources sur les plateformes cloud, à travers la caractérisation de la

charge de travail, l'étude de la spécificité des applications déployées, et l'évaluation d'algorithme de placement d'opérateurs de traitement. En second lieu, elle détaille une architecture Big Data de bout en bout idéal pour les workflows de l'industrie automobile. L'algorithme étudié ne peut pas s'adapter au flux de données variable dynamiquement. Nous pensons néanmoins qu'un tel niveau d'adaptabilité pourraient être utiles, notamment pour préserver un haut niveau de service avec un bon flux de données et moins de temps d'arrêt en raison de plusieurs compilations.

MOTS-CLÉS: architecture Big Data, optimisation de requêtes, Cloud Computing, écosystème Hadoop, Stream Computing, véhicules connectés

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Associate Professor Ahmed Mostefaoui, my research supervisor, and François Dessables, my company supervisor, for their patient guidance, enthusiastic encouragement, and valuable critiques of this research work. My grateful thanks are also extended to Mr. Eric Jacquet for his help in doing the stress tests and data analysis, to Mr. Amir Haroun, who helped me strengthen my knowledge in stream computing engines, and to Ms. Crisitina Mateos for her support in diverse technical issues.

I would also like to extend my thanks to the team members and professors of the AND team (Algorithmique Numérique Distribuée) for the positive work environment and the pleasant moments we shared together. I want to express my gratitude to all the Ph.D. students at the Femto-ST lab, with whom I spent beautiful moments during the past years.

My beyond admiration and thanks to all the team members at Stellantis (f.k.a Groupe PSA) with whom I had the opportunity to work and share my workplace. This vibrant, diverse, and thoughtful team supported and challenged me, making me a better researcher and a confirmed data engineer. Also, I would like to thank the Stellantis doctoral students animation team for providing me with the resources necessary for managing the program and the moral and mental support throughout the thesis.

Finally, I wish to thank my parents, family, and friends for their support and encouragement throughout my study.

CONTENTS

I

INTRODUCTION

1

INTRODUCTION

This thesis focuses on advancing the state of the art of automotive Big Data, with a specific focus on resource allocation in stream processing applications and on the general IT architecture of the Big Data Analytics framework. The presented research was carried out in the department of informatics and complex systems (DISC) of the Femto-ST laboratory with the French automobile manufacturer's collaboration, Groupe PSA (refer to Appendix A). This chapter provides an introduction to the work done in this thesis. It addresses the general context, the small set of challenges studied in this work, and then briefly presents this thesis's contributions.

1.1/ GENERAL INTRODUCTION

Big Data has become a central subject for various sectors, scientific disciplines, and the whole of society. This popularity is due to the ability to produce, capture, distribute, process and analyze vast quantities of diverse data with almost universal value, and radically change how people live and use modern technology, how businesses work, and how science can be conducted. Different industries such as banking, automotive, manufacturing, or healthcare will significantly benefit from better and faster data processing, as demonstrated by current developments in the industry such as "Internet of Things" and "Industry 4.0." Data-driven approaches to research using Big Data technologies and analysis are becoming increasingly popular in geosciences, astronomy, or life sciences, for example. Users using social media, web tools, and smart devices spend growing amounts of time online, creating and consuming vast quantities of data, and targeting tailored content, ads, and reviews. Most of the future Big Data-related advances are still in an early stage. However, if the numerous technical and application-specific challenges in the management and use of Big Data are successfully tackled, there is great hope. Some of the technological challenges have been correlated with different "V" attributes, expressly Volume, Variety, Velocity, and Veracity. Other issues relate to the security of confidential and sensitive data to maintain a high degree of privacy and the ability to translate the vast

amount of data into valuable insights or better activity. The net result of all of these developments is that the existing technical environment for Big Data is not yet established. However, there are several potential approaches within the Hadoop ecosystem but also within the product range of various database vendors and other IT companies (such as Google, IBM, Microsoft, Oracle). The early version of Hadoop was highly efficient only to attain its limits in various fields, e.g., to help the processing of fast-changing data such as data streams or process highly iterative algorithms, e.g., for graph processing or machine learning. Besides, the Hadoop environment was essentially decoupled from the common approaches to data processing and analysis, based on relational databases and SQL. These factors have resulted in many additional components within the Hadoop ecosystem, including general-purpose processing frameworks such as Apache Spark and Flink and specialized components, e.g., for graph data, data sources, or machine learning. Also, several methods now exist to combine Hadoop-like data processing with relational database processing ("SQL on Hadoop").

A principal facilitator for the Big Data tendency is the powerful and affordable computing platforms that allow fault-tolerant storage and processing of petabytes of data within large computing clusters, usually fitted with thousands of processors and terabytes of memory. Web giants such as Google and Amazon pioneered such infrastructures, but open-source framework software such as the Hadoop ecosystem made that possible. Originally, Hadoop was composed of a few core components, particularly its distributed HDFS file system and the MapReduce framework for the relatively simple development and execution of highly parallel applications to process large amounts of data on cluster infrastructures.

The idea of an Intelligent Transportation System (ITS) was implemented to efficiently manage traffic, improve road safety, and conserve our green environment. ITS applications are becoming more data-intensive nowadays, and their data are represented using "5Vs of Big Data." Big data analytics, therefore, need to be implemented to exploit such data fully. Internet of Vehicle (IoV) links the ITS computers to cloud computing centers where data processing is carried out. The conceptual idea of Vehicular Ad Hoc Networks (VANETs) was introduced over a decade ago, where vehicles fitted with wireless communication devices would form networks. In a VANET, communication between vehicles (V2V) or between vehicles and infrastructure networks (V2I) can be possible. The primary goal of VANETs is to improve road safety. Nevertheless, the idea of VANETs transformed into the Internet of Vehicles (IoV) to meet the growing and evolving requirements of ITS's applications. Connected vehicles serve as hubs in the IoV world for sensing and tracking traffic congestion status, road conditions, and environmental pollution levels. Advanced driver assistance systems and eventually self-driving applications require many computing and communication skills to succeed in their computationally intensive and latency-sensitive tasks. As reported by Intel, multiple sensors would be required to collect the

necessary information on the driver's continually changing environment and the ability to fuse the data (approx. 1 GB/s) to make safe decisions. Meanwhile, passenger infotainment technologies were growing from Internet platforms such as social media services and email to online video sharing and multiplayer gaming and augmented and virtual reality applications. Thus, the generated ITS data in the IoV environment is voluminous, and the processing of these data requires big data analysis.

1.2/ FOCUS OF THE THESIS

As illustrated in the previous section, the challenges of Big Data are numerous and take different shapes and forms, let alone the specific constraints added by the Intelligent Transportation System (ITS) to the equation. From data transmission and communication protocols to highly parallel computing platforms and, when required, latency-sensitive engines, there is plenty of opportunities that the scientific community can grasp and explore. The work carried out in this thesis focused on, paradoxically, the big picture of the Big Data's solution architecture specialized for the automotive industry's requirements, and zoomed into the essential task of allocating physical resources to processing applications.

1.3/ MAIN CONTRIBUTIONS

The main contributions in this dissertation fall within the aforementioned processing chain of the data produced by the Internet of Vehicles (IoV). They meet the above requirements while considering the resources constraints present in the Cloud computing environment and its management's subtleties. Our research's principal contributions—aside of the engineering related-tasks and complementary work accomplished beside Groupe PSA team—are summarized as follows:

1. First, we review a basic and essential process in Stream Processing Engine, a resource allocation algorithm. A streaming application can be modeled as a directed graph where vertices are operators' instances, and edges are data streams (i.e., continuous series of tuples). The most challenging aspect of implementing streaming applications is figuring out how to map operators' graphs to available physical resources to improve performance (increasing the throughput and reducing processing time). We addressed this problem by demonstrating that relying on inherent data parallelism does not always result in the best results for all applications. Through a real-world application (the Eco-Driving service), we demonstrated that a mapping focused on in-depth analysis of the target application's details and

infrastructure features could dramatically improve application efficiency, resulting in a noticeable improvement.

2. Second, we rethink Big Data architecture and design an end-to-end architecture that meets the needs of today's data-intensive applications. Connected vehicles (CVs) can now capture up to 170 different types of data (e.g., speed, temperature, and fuel consumption) from onboard built-in sensors and send it to infrastructure in real-time, typically through 4G/5G wireless communications. This reality opens up numerous possibilities for developing new and creative telematics services, such as driver protection, customer experience, efficiency and reliability, location-based services, dealer services, and infotainment, to name a few. By the end of 2025, it is estimated that there will be approximately 2 billion connected cars on the road, each capable of producing up to 30 terabytes of data every day. The underlying data management infrastructure is put under much pressure when managing big data, whether in real-time or in batches. In this paper, we look at the big data platform used by real CVs, specifically the one used by Groupe PSA[1]. We emphasize why the Hadoop framework is no longer the *de-facto* big data solution by presenting technologies and open-source products used within various platform components to collect, store, process, and, most importantly, exploit big data.

1.4/ PUBLICATIONS

CONFERENCE PAPER

- ***Anthony Nassar***, Ahmed Mostefaoui, and François Dessables. *"Improving Big-Data Automotive Applications Performance Through Adaptive Resource Allocation"*. In **IEEE Symposium on Computers and Communications (ISCC)**, pp. 595-601, Barcelona 2019.

SUBMITTED JOURNAL PAPER

- Ahmed Mostefaoui, Mohammed A. Merzoug, Amir Haroun, ***Anthony Nassar***, François Dessables. *"Automotive Bid-Data Architecture: Experimental Feedback From Groupe PSA"*. In **IEEE Transactions on Vehicular Technology**. Submitted in March 2021.

[1] Groupe PSA: a French car manufacturer. https://www.groupe-psa.com/en/

1.5/ DISSERTATION PLAN

The rest of this dissertation is organized as follows: Chapter 2 discusses the scientific background of IoV in general. Chapter 3 examines the Hadoop Ecosystem that enabled the exploitation of data by scientists. Chapter 4 zooms in to one of the most important bricks in the Big Data ecosystem. The brick that is mainly focused on processing the continuous flow of data, i.e., stream processing and stream engine. Chapter 5 presents this dissertation's first contribution, namely an automated resource allocation algorithm for streaming applications. Chapter 6 introduces an end-to-end architecture for dealing with Big Data automotive applications. Chapter 7 concludes the work that has been done in this thesis.

II

BACKGROUND

2

Intelligent Transportation System, VANET and Internet of Vehicles

*"An **intelligent** hell would be better than a stupid paradise."*

Victor Hugo, Ninety-Three

2.1/ Introduction

The emergence of intelligent transportation systems is mainly due to major technological advances in the fields of wireless communication protocols and real-time and on-board systems. Very promising, these systems make it possible to offer a wide range of new applications, new communicating vehicles and to define an ecosystem of diversified mobilities. The inherent characteristics of these systems pose new challenges in terms of communication protocols and architectures. This chapter provides an introduction and overview of Intelligent Transportation Systems, vehicle networks and the Vehicle Cloud and the various relationships between them. The chapter also examines the architectures and types of communication that govern these networks and discusses the different objectives and applications that characterize these technological concepts.

2.2/ Vehicular Ad Hoc Network (VANET)

The conceptual idea of Vehicular Ad Hoc Networks (VANETs) originated more than a decade ago and has since been a highly active research area [1]. VANETs' basic concept

treats vehicles as mobile nodes that can interact in order to establish a network [2]. Because of mobility constraints, VANETs are considered to be conditional networks where their efficiency is influenced by vehicle density and distribution and other factors such as lousy driver conduct and high-rise buildings. The vehicles are further called unreliable, temporary, and random nodes. Therefore, VANETs can not guarantee the viability of large scale consumer applications/services [3]. VANETs are, therefore, best suited to limited-scale applications involving ad hoc services such as preventing accidents or notifying drivers of road hazards.
Nevertheless, new connectivity criteria for VANETs are increasing due to the growth of the Internet of Things (IoT) technology and the rise in the number of Internet-connected vehicles. Another drawback of VANETs is their limited capacity to process information that is collected by themselves and actors around them (such as mobile devices and sensors) [2]. To meet the new ITS requirements, vehicles will act as a smart multi-sensor platform with IP-based Internet networking, multiple communication systems, powerful computational components, and the ability to communicate with other vehicles and ITS computers [4].

2.2.1/ The evolution of IoV

The development of the scientific definition of VANETs culminated in the creation of the concept of the Internet of Vehicles (IoV) [5]. As a special case of IoT, IoV thus has distinctive characteristics and unique specifications to support the intelligent transport systems. An IoV is characterized as a platform that fully realizes the integration and exchange of information between people, vehicles, items, and the environment [6]. IoV's primary goal is to increase transportation safety and efficiency, boost city service quality, save the environment, and ensure that people are happy with the transportation system services. Like VANETs, IoV combines vehicle intelligence with vehicle networking, resulting in smart networks with communication and computing capabilities that provide large scale transport services [3].
In the IoV climate, since vehicles have permanent Internet links, they can provide information for the different categories of ITS applications (i.e., road protection, traffic management and control, and infotainment). As a result, the exchange of information between sensors and electric actuators, road infrastructures, and vehicles, as well as drivers and passengers is allowed [2]. IoV gathers a huge amount of data from a broad region of various systems, which is in line with the idea of big data heterogeneity [7].
With the significant advantages IoV has over VANETs, it opens up several new possibilities. IoV gives users, communities, and economies several different benefits. Cisco IBSG Automotive and Economics activities expected that the advantages of using IoV technology for each vehicle could exceed $1,400 USD per year. Also, reductions in traffic

congestion and enhancements to road safety will produce substantial financial savings in the public health sector. Besides, the use of real-time traffic solutions that connected vehicles can lead to less time spent in traffic jams and improved productivity. More specifically, service providers can find opportunities through IoV deployment to implement new transportation technologies such as real-time traffic monitoring, location of parking lots, and customer support based on the location [8].

2.2.2/ Existing IoV System Architectures

Throughout the years, many attempts have been taken to develop an ideal IoV architecture that can serve as many IoV scenarios and applications as possible. In this paragraph, we review the latest architecture of all sizes published in academic journals. Table 2.1 lists the proposed architectures for IoV applications and gives a summary of the composition of each.

An architecture of three layers is defined in [9], which describes the different interactions between IoV system technologies. The first layer consists of all the sensors of the vehicle that gather environmental data and detect certain essential events such as traffic circumstances, driving habits, and ambient environmental conditions. The second layer is for communications that support various wireless modes, Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), Vehicle-to-Sensor (V2S), and such as Vehicle-to-Pedestrian (V2P). Seamless access is assured across the networking layer to various networks such as IEEE 802.11p, IEEE 802.15.4, Wi-Fi, Bluetooth, GSM, and LTE. The third layer has the expertise of IoV intelligence and is responsible for making decisions under challenging circumstances (e.g., hazardous road conditions and accidents). This layer includes tools for analytics as well as the information gathered for storing and analyzing big data.

In [5], they implemented a layered IoV protocol stack and architecture. The architecture is a five-layered one. The first layer is the layer of perception that is defined by the different forms of personal computers, RSUs, vehicles, sensors, and actuators. The second layer is the communication layer that includes a virtual network containing different network technologies such as Wi-Fi, WAVE, 4G/LTE, and satellite networks. The artificial intelligence is the third layer reflecting the virtual cloud infrastructure, where the obtained information is stored, processed, and analyzed. The fourth layer is the layer of application that contains smart ITS applications. The last layer is the business layer, which represents IoV's operational management module. Besides, the author built a protocol stack based on the proposed five-layer architecture to coordinate the current protocols. There are three planes in the built protocol stack, including management, operation, and security.

Table 2.1: Summary of existing IoV architectures

Year	Architecture	Characteristics
2016	Golestan et al. [9]	3 layers (client, connection and cloud)
2016	Kaiwartya et al. [5]	5 layers (perceptron, coordination, AI, application and business)
2017	Yang et al. [10]	4 layers (environment sense and control, network access and transport, coordination computing control and application)
2017	Contreras et al. [11]	7 layers (user interface, data acquisition, data filtering and pre-processing, communication, control, management and security)
2019	Sherazi et al. [12]	3 layers (client, communication and cloud)

In 2017 Fuyung et al. [10] recently introduced a four-layer IoV architecture. The architecture is distinct from the architectures of the past. The first layer is the sensing and control layer of a vehicle network system. Environmental sensing is the basis for the identification of IoV services, such as autonomous vehicle services, smart traffic, and vehicle details. Vehicle control and the object traffic system are the basis for the implementation of IoV services. The second layer is the layer of network access and transport, and its crucial role is to accomplish network access, data collection, data analysis, and data transfer. The third layer is the power layer for synchronization in computation. This layer presents IoV applications with the network-wide capability of human-vehicle-environment coordinating computation and control, such as data processing, swarm intelligence, and resource management. The last layer is the implementation layer for delivering facilities to satisfy the human-vehicle-environmental coordinating systems specifications.

Another group of researchers [11] suggested an IoV model architecture of seven layers. Their architecture enables the direct interconnection of all network components and the dissemination of data in an IoV environment. The first layer is the user experience layer that provides direct contact with the driver to organize all driver alerts through a management interface. It also selects the best display feature to assist and decrease driver disturbances, depending on the current situation. The data collection framework is structured to collect data from various sources, which include internal sensors for the car, such as a global positioning system (GPS), traffic signs, road signals. The third layer is the layer for pre-processing and filtering the data. Throughout this layer, the data gathered are pre-processed to reduce network traffic by removing irrelevant transmission information. Transmission requirements are based on a service profile generated for the vehicle having subscribed or disabled services. The communication layer is designed to select the network most appropriate for sending the information. Selection criteria such as congestion and degree of QoS across the various networks available, information relevance, privacy, and protection, among others, are considered. The fifth layer is the layer of reg-

ulation and management. This layer integrates and manages numerous network service providers that are within the IoV ecosystem globally. Different tasks and procedures are implemented to handle the collected information, e.g., traffic control, traffic engineering, packet inspection. The sixth layer, which is the business layer, is aimed at processing the enormous amount of information. It includes infrastructure for Cloud computing.
This layer also includes functions to store, process, and analyze information obtained from lower layers. Decision-making is focused on a statistical analysis of the results.
Strategies to assist in the implementation of business models based on data use in applications and statistical analysis were established. The protection layer is a transversal layer that stretches to provide direct communication through other layers. Inside the IoV architecture, protection functions (e.g., data authentication, privacy, non-repudiation, and confidentiality, access control, availability) can be found to help mitigating approaches to resolve different types of security attacks.

In [12], the authors proposed a heterogeneous architecture which uses different communication protocols. The architecture follows a three-layered approach to simplify the functionalities of various components.
The client layer at the bottom includes intra-vehicle and inter-vehicle communications (e.g., communication between different sensor nodes within a vehicle). It is also responsible for allowing IoV to express and maintain a trustworthy Cyberspace identity. The communication layer deals with the interconnectivity within a network of different network components and the integration of other accessible networks within the vehicle setting. Similarly, it is primarily the responsibility of the cloud layer to allow all IoV resources and applications. It also provides multiple cloud-based services, such as virtualization, data storage, and real-time interactions between different network entities.

2.3/ INTELLIGENT TRANSPORTATION SYSTEM

Intelligent Transportation Systems are a global phenomenon, attracting worldwide interest from transportation professionals, automotive industry, and political decision-makers. ITS applies advanced communication, information, and electronics technology to solve transportation problems such as traffic congestion, safety, transport efficiency, and environmental conservation, as represented in Figure 2.1.

The goal of ITS, we might claim, is to take advantage of the correct technology to build ”smarter” highways, vehicles, and users. Since the ’30s, ITS has been around and slowly creeping into our lives. The significant developments in ITS have been made in Japan, Europe, and the United States. They have gone through three phases [14]: preparation (1930-1980), feasibility study (1980-1995), and product development (1995-present) as represented in Figure 2.2.

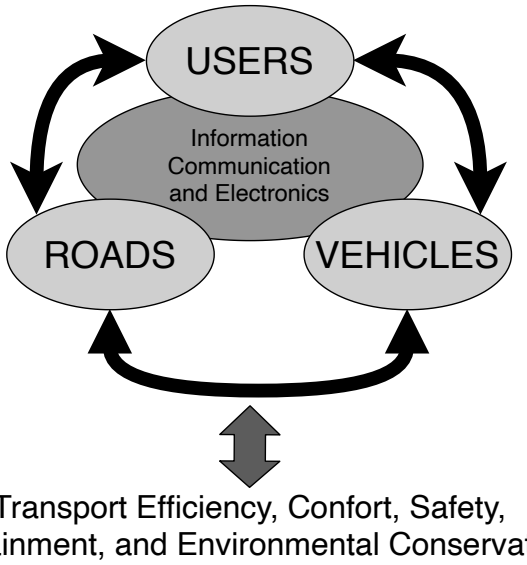


Figure 2.1: ITS Conceptual Model. Taken from [13].

2.3.1/ History of ITS

A./ Preparation (1930-1980)

Preparation is the first phase of ITS development. At the same time, IT technologies had not yet matured enough, and the focus was all about improving the transportation infrastructure by building new roads and bridges. The first ”original” ITS system was the electric traffic signals implemented in 1928. At the GM pavilion of the 1939 world fair in New York, a concept for Automated Highway Systems (AHS) was presented. However, the ITS movement did not take root until the 60s, when the first computer-controlled traffic signals in the US appeared. From the late 60s up to 1970 in the US, the Electronic Route Guidance Systems (ERGS) was developed, which used a two-way road vehicle communications to provide route guidance. During the 70s, the Comprehensive Automobile Traffic Control System (CACS) and the Autofahrer Leit und Information System (ALI) were developed in Japan and Germany respectively, which are dynamic path control mechanisms focused on real traffic conditions [15]. This decade was also crucial for ITS due to the microprocessor introduction and the start of GPS development. These technologies are now major components of many ITS systems; nevertheless, they were not associated with ITS.

B./ Feasibility study (1980-1995)

This period is marked by an explosion in industry-subsidized construction projects in Europe, Japan, and the US. These programs resulted from the underlying concepts and es-

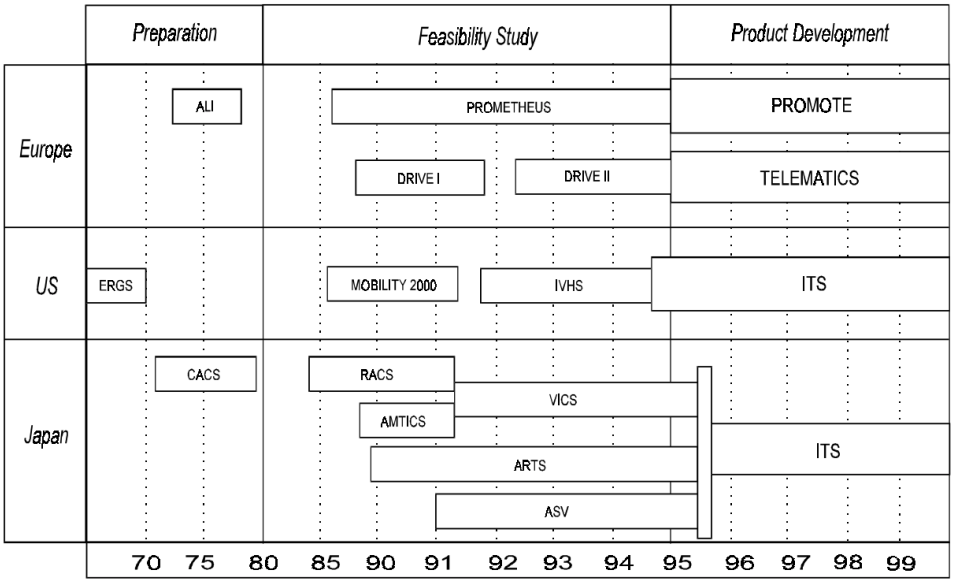


Figure 2.2: ITS Development Chronology in Europe, US and Japan. Taken from [13].

sential ITS technologies developed during the previous phase. In Europe, governments, companies, and universities of 19 countries established the PROMETHEUS (Program for European Traffic with Efficiency and Unprecedented Safety) project. Several ITS technologies were developed in this program between 1987 and 1994. In the 80s, the test vehicle VaMoRs was demonstrated in Munich [16]. Two forward-looking TV cameras were used in this prototype for an automatic lane and the road following. In the 90s, a group led by Daimler-Benz developed a prototype to keep the vehicle in the lane and to keep a safe distance with its surroundings. This work was done under the VITA II project [17]. This program was followed by DRIVE (Dedicated Road Infrastructure for Vehicle Safety in Europe) for the development and test of the communication system for driver assistance and traffic management [18]. Around the late 80s in the United States, the Mobility 2000 study team laid the groundwork for the establishment of the IVHS America (Intelligent Vehicle Highway Systems) [19], which is a public-private forum for consolidating national ITS interests and promoting international cooperation in ITS. In 1994 the USDOT (United States Department of Transportation) changed the name from IVHS to ITS America (Intelligent Transportation Society of America). A key project, AHS (Automated Highway System), was conducted by NAHSC (National Automated Highway System Consortium) composed of the US Department of Transportation, General Motors, University of California, and other institutions [20]. Different fully automated test vehicles have been demonstrated on California highways under this initiative. In Japan, in the 80s, several programs have been launched, namely RACS (Road Automobile Communication System) [27] by the Ministry of Construction and AMTICS (Advanced Mobile Traffic Information and Communication System) [21] by the National Police Agency. In the 90s, with the combined efforts between the Ministry of Posts and Telecommunications and work on standardiza-

tion projects, the combination of those two projects was made possible. It resulted in the VICS (Vehicle Information and Communication System). A VICS terminal offers a locator to view the vehicle coordinates on the map screen and allows contact with ground stations to obtain traffic conditions for route planning. Representatives from academia and industry coordinated VERTIS (Vehicle, Road, and Traffic Intelligent Society). That society conducted various ITS-related activities, namely information exchanges with its European and American counterparts, ERTICO, and ITS America. In 1996, the Ministry of Construction and twenty-one major companies, namely Honda, Toyota, Mitsubishi, and Nissan, formed the Advanced Cruise-Assist Highway System Research Association and implemented on a highway various fully-automated vehicles [22].

C./ PRODUCT DEVELOPMENT (1995-PRESENT)

The previous step concentrated on establishing a technological base, with high-level ITS functions, and this aim was successfully achieved. A unified policy was adopted around the mid-1990s to deal consistently and harmoniously with ITS. This policy led to the present phase, dealing with the creation of possible products. Several projects were developed. The Chauffeur project, by Daimler-Benz and research institutes, is one example in Europe. Their objective was to give trucks the ability to follow automatically another truck driven by a human. The primary focus of ITS systems in the US moved to large-scale integration and implementation by the late '90s.

2.3.2/ CHALLENGES OF ITS

First, as the number of cars on the road increases, congestion has become an increasingly significant problem worldwide. For instance, in early 2010, Beijing, China, had a total of 4 million vehicles and produced 800 000 more in that year. Congestion will lead to fuel consumption, air pollution, and difficulties in the implementation of public transport plans [23]. Those problems can also raise the risk of heart failure, as a medical study states [24].

Second, with the expansion of transport systems, especially in several developing countries, the risks of accidents are increasing [25]. Malta et al. [26] found out that almost three-fourths of all traffic incidents can be ascribed to human error. The U.S. Federal Highway Administration reported that traffic incidents that occurred in cities constitute about 50 to 60 percent of all congestion delays [27]. There is no doubt a need to reduce traffic accidents and detect accidents once they have occurred in order to minimize their impact.

Third, in many countries, soil resources are often limited. Building modern infras-

tructures such as highways and freeways, therefore, is challenging. Following the terrorist attacks in New York City on September 11, 2001, the effectiveness of transportation networks is increasingly tied to the capacity of a nation to manage emergencies (e.g., mass evacuation and enhancement of security) [28, 29, 30]. A country's competitiveness, economic strength, and productivity depend heavily on its transport systems [31].

2.3.3/ CURRENT SOLUTIONS

The implementation of new transport policies can solve some of the problems mentioned above. For instance, the city government of Beijing, China, placed a ban on car owners based on odd/even license plate numbers during the 2008 Beijing Olympics to keep 50 percent of private cars off the streets. This approach has undoubtedly, to some extent, eased congestion and the problem of air pollution. Overall, the approach works for special events, but in nominal circumstances may not be appropriate.

Another approach is to add additional infrastructure by constructing new roads and improving existing infrastructure, such as road widening. However, this approach can be costly and demanding for the exploitation of already minimal land resources.

The third approach is to maximize the use of the current transportation infrastructure by evaluating the data obtained from a broad range of auxiliary instruments, such as inductive loop detectors, sensors, receivers based on the Global Positioning System (GPS), and microwave detectors. Ideally all three methods would be mutually compatible.

Note that data can currently be processed into useful information and can also be used to generate new functions and services in intelligent transportation systems (ITS). For instance, GPS data may be used to evaluate and forecast traffic user behavior, which is a feature that is not entirely used in traditional ITS. Also, gathered vehicle data can calculate a score, determine an ecological profile of the drivers, and help them improve their environmental impact.

2.4/ CONCLUSION

Intelligent transportation networks result from significant technological advancements in wireless communication protocols and real-time and onboard systems. These fascinating systems allow for a wide range of new applications, new interacting vehicles, and the development of a diverse mobility ecosystem. In terms of communication protocols and architectures, the inherent characteristics of these structures present new challenges. This chapter gives an overview of Intelligent Transportation Systems, vehicle networks, the Vehicle Cloud, and their different relationships. The architectures and types of com-

municative systems are also examined in this chapter. The architectures and modes of communication that control these networks are also discussed, and the various goals and applications that define these technical concepts.

This chapter has set the background of the Big Data domain applied to the vehicular domain. In this thesis, we work on detailing the backend architecture, that is, the Vehicle Cloud of Groupe PSA that leverages vehicular data to provide several valuable services for the community.

3

General-Purpose Processing Framework

*"If one is master of one thing and understands thing well, one has at the same time, **insight** into and understanding of many things"*

Vincent Van Gogh

3.1/ The Hadoop system: The origins

Back in 2004, Google published a paper [32] detailing a simple and powerful programming model that allows straightforward development of scalable parallel applications to process vast amounts of data on large clusters of commodity machines. Specifically, the implementation mentioned in the original paper is designed mainly to achieve high performance on large commodity PC clusters. One of the key benefits of this strategy is that it isolates the application from the specifics of running a distributed program, such as data storage problems, scheduling, and fault tolerance. The computation consists of having a set of key/value pairs as input and produces a set of key/value pairs as output. The user expresses the computation using two elementary functions: *Map* and *Reduce*. The *Map* function takes a pair of inputs and creates a set of intermediate key/value pairs. All intermediate values associated with the same intermediate key A are grouped by the MapReduce system and transferred to the *Reduce* function. The *Reduce* function gets and merges an intermediate key A with its set of values. Usually, only zero or one output value is generated by the invocation of the Reduce function. This model's key benefit is that it enables massive computations as the primary mechanism for fault tolerance to be conveniently parallelized and re-executed.

3.1.1/ HADOOP / MAPREDUCE

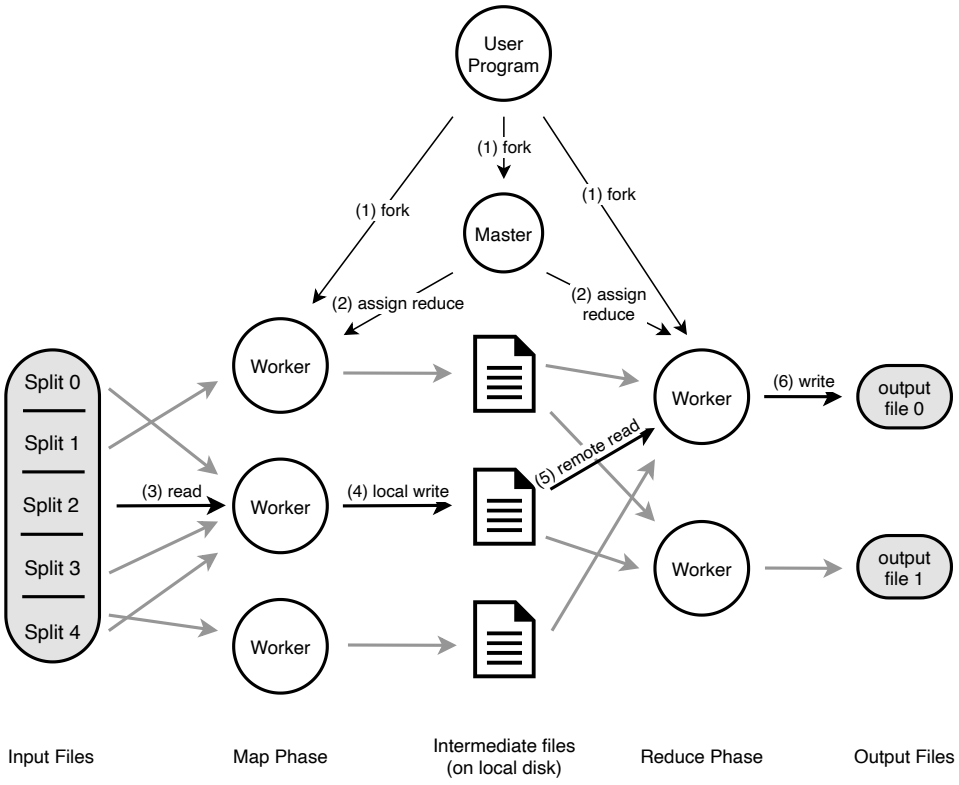


Figure 3.1: An overview of the execution flow of a MapReduce job

Hadoop is an open-source Java library [33] that supports distributed data-intensive applications by implementing the MapReduce framework. It has been commonly used for development purposes by a significant number of business firms. The MapReduce job's Map function calls are practically spread through multiple machines by automatically partitioning the input data into N splits. Different machines can process the input splits in parallel. Reduce function calls are distributed by partitioning the intermediate key space into P pieces using a partitioning function (e.g., hash(key) mod P). The user determines the number of partitions (R) and the partitioning function. Figure 3.1 illustrates an example of the overall flow of a MapReduce procedure that passes through the following action sequence:

1. The MapReduce program input data is split into N pieces, and several instances of the program are started on a cluster of machines.

2. One instance of the program is selected to be the master copy, while the rest are considered to be staff assigned by the master copy to their job. Mainly N map tasks exist, and P reduces tasks to be assigned. The master selects idle workers and assigns each or more tasks to the map and reduce tasks.

3. A map task is assigned to a worker that processes the contents of the respective input split and generates key/value pairs from the input data and passes each pair to the user-defined Map function. In memory, the intermediate key/value pairs generated by the Map feature are buffered.

4. The buffered pairs are regularly written to a local disk and partitioned by the partitioning function into P regions. The addresses on the local disk of these buffered pairs are transferred back to the master, responsible for transmitting these addresses to the reduced workers.

5. When the master notifies a reduced worker of these addresses, it reads the buffered data from the map workers' local disks, which are then sorted by the intermediate keys such that all instances of the same key are grouped. Sorting operation is essential because several different main maps usually reduce the task to the same.

6. The reduction worker transfers to the Reduce feature of the user the key and the corresponding intermediate values set. For this reduced partition, the output of the Reduce function is appended to a final output register.

7. The master program wakes up the user program when all the map tasks and reduced tasks have been completed. At this point, MapReduce's invocation in the user program returns the program's control to the user code.

The Hadoop project was introduced as an open-source Java library supporting data-intensive distributed applications and cloned Google's MapReduce framework [32]. The Hadoop architecture consists, in theory, of two key components: the Hadoop Distributed File System (HDFS) and the MapReduce programming model. In particular, HDFS provides the basis for distributed storage of big data, which distributes data files into data blocks and stores them in different computer cluster nodes to allow efficient parallel processing of data.

3.1.2/ HADOOP ENHANCEMENTS

In a heterogeneous environment with several different storage systems, the straightforward implementation of MapReduce is beneficial for managing data processing and data loading. It also offers a versatile structure for more complicated function execution that can be explicitly supported in SQL. However, there were some drawbacks to this simple architecture. In the following section, we analyze some research efforts that have been carried out by presenting different changes to the MapReduce system's basic implementation to overcome these limitations.

A./ Supporting Iterative Processing

The simple MapReduce system does not directly support iterative data analytic applications. Instead, by issuing several MapReduce jobs manually and orchestrating their execution using a driver program, programmers could implement iterative programs. The HaLoop system [34] is designed to enable iterative processing of the MapReduce framework by expanding the two key functionalities of the basic MapReduce framework:

1. Caching the first iteration of the immutable data and then reusing them in subsequent iterations.

2. Caching the reducer's outputs, which makes it more successful to search for a fixpoint, without an additional MapReduce task.

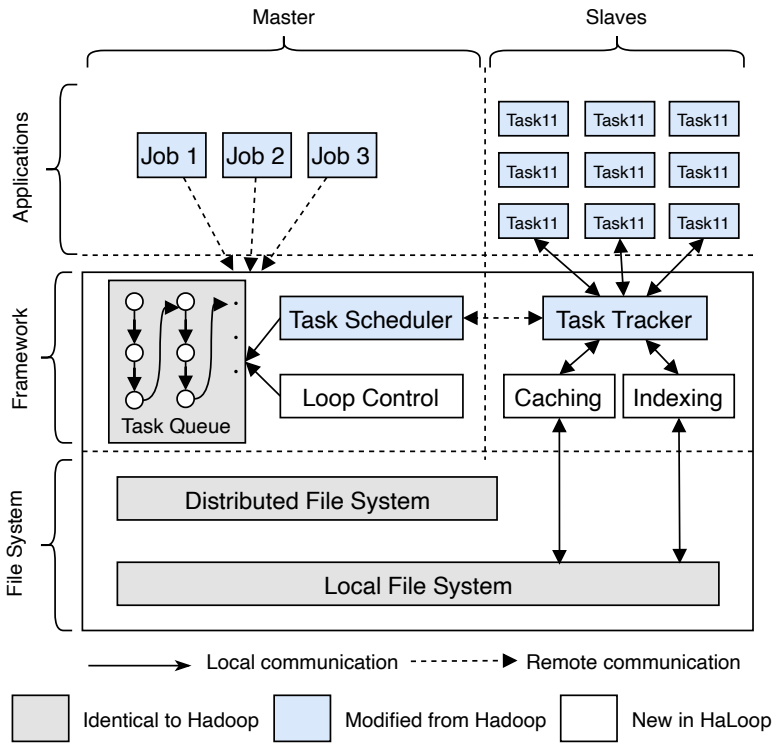


Figure 3.2: An overview of HaLoop architecture [34]

Figure 3.2 demonstrates HaLoop's architecture as an updated version of the basic structure of MapReduce. HaLoop essentially relies on the same file system and has the same task queue structure as Hadoop, but the task scheduler and task tracker modules are changed and the loop control, caching, and indexing modules are added to the architecture. The task tracker monitors the execution of tasks and monitors caches and indices on the slave node and redirects the cache and index accesses to the local file system for each task.

B./ Processing Join Operations

The MapReduce framework's main drawback is that it does not support joining multiple datasets into one mission. With additional MapReduce workarounds, however, this can still be done. For example, users can map and reduce one data set on the fly and read data from other datasets. The Map-Reduce-Merge model [35] has been implemented to enable the processing of multiple datasets to tackle the restriction of the additional processing requirements for conducting joint operations in the MapReduce system. The structure of this model is shown in Figure 3.3.

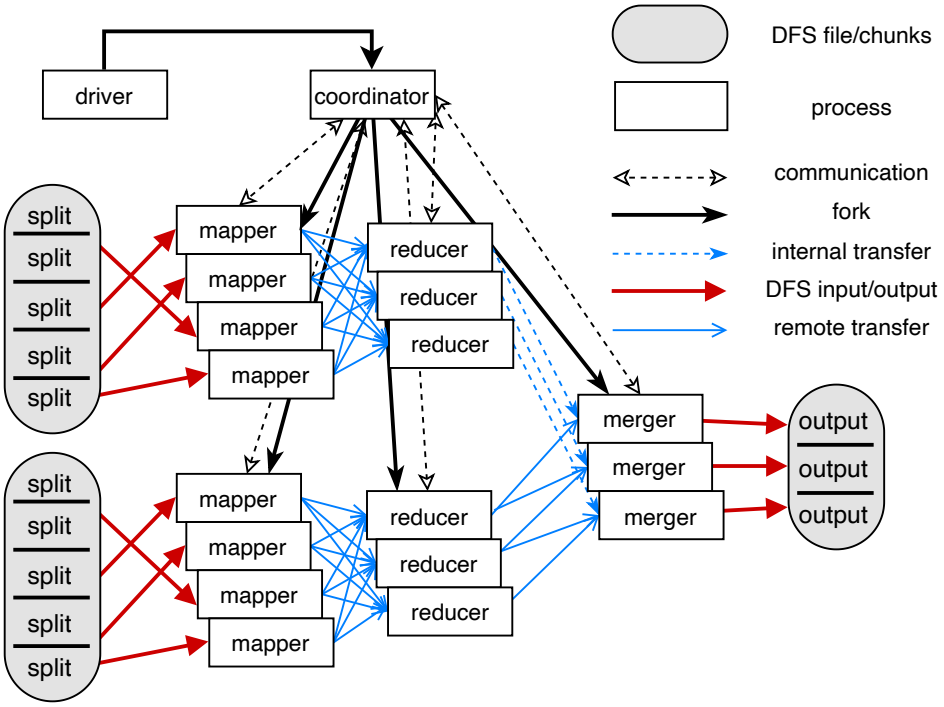


Figure 3.3: An overview of the Map-Reduce-Merge framework [35]

The main differences between this framework's processing model and the original MapReduce are that a key/value list is created from the Reduce function instead of just the values. This change is made because the merge function requires keys to arrange input datasets (partitioned, then either sorted or hashed), and these keys must be passed into the function to be merged. The reduced result is, contrarily, final in the original setting.

C./ Data and Process Sharing

The use of an analytical query processing infrastructure (e.g., Amazon EC2) can be directly mapped to monetary value with the advent of cloud computing. There may be several possibilities for sharing their work's execution, taking into account that various MapReduce workers will perform similar work. This sharing will also minimize the total

amount of work, resulting in a decrease in the monetary charges incurred when using the production infrastructure services. The MRShare system [36] has been described as a sharing mechanism optimized to turn a batch of queries into a new batch that will be more effectively executed by combining jobs into groups and evaluating each group as a single query. They presented an optimization problem based on a given cost model that aims to extract the optimal grouping of queries to prevent redundant work from being performed and thus result in significant savings in both processing time and money.

D./ Effective Data Placement

The purpose of the data placement policy in the basic implementation of the Hadoop project is to achieve good load balancing by spreading the data uniformly across the data servers, regardless of the planned usage of the data. For most Hadoop applications that access only a single file, this simple data placement policy works well. However, with tailored techniques, some other applications process data from multiple files to get a significant performance boost. In these applications, the absence of data co-location increases data shuffling cost, increases the network's overhead, and decreases data partitioning performance. CoHadoop [37] is a lightweight Hadoop extension designed to allow related files to be co-located at the file system level while maintaining the properties of good load balancing and fault tolerance at the same time.

3.2/ Hadoop Ecosystem

The Hadoop ecosystem has been expanded throughout the years with different components (see Figure 3.4). For example, when performing tasks with a different workflow (e.g., joins or n stages), the MapReduce programming model's over-simplicity and its dependency on a rigid one-input and two-stage dataflow lead to situations in which inelegant workarounds are needed. To support SQL-on-Hadoop with popular relational database concepts such as tables, columns, and partitions, the *Hive* project [38] was launched. It supports queries represented in the Hive Query Language (HiveQL), an SQL-like declarative language representing a subset of SQL92, and can therefore be easily understood by anyone familiar with SQL. These queries are compiled into Hadoop jobs automatically. Hive abstracts away underlying MapReduce jobs and returns HDFS in the form of tables (not HDFS). *Pig* was introduced as a high-level scripting language (Pig Latin) that allows complex data transformations to be written. It pulls, cleans, and puts unstructured/incomplete data from sources into a database/data warehouse. Although Hive queries the data warehouse to perform analysis, Pig performs ETL in a data warehouse.

Impala[1] is another open-source project developed by Cloudera to provide an SQL query engine that runs natively in Apache Hadoop with massively parallel processing. It uses the standard Hadoop infrastructure components (e.g., HDFS, HBase, YARN) and can read most of the file formats used commonly (e.g., Avro, Parquet). Therefore, the user can query the data stored in the Hadoop Distributed File System (HDFS) using Impala.

Another part that has been implemented as an open-source project that supports large-scale graph processing and clones the implementation of the Pregel framework by Google [39] is *Apache Giraph*. Giraph runs graph processing jobs on Hadoop as map-only jobs and uses HDFS to input and output data. *Apache Hama*[2] is another deployment project based on BSP that, like Giraph, is designed to run on top of the Hadoop infrastructure. It focuses, however, on general BSP computations and not just on the processing of graphs. It includes algorithms for matrix inversion and linear algebra, for instance. Another type of application that is iterative is machine learning algorithms. The *Apache Mahout*[3] project was built on top of the Hadoop framework to develop scalable machine learning libraries. *Apache Tajo*[4] is another distributed Apache Hadoop data warehouse framework designed for low-latency and flexible ETL processes for ad-hoc queries. Tajo can process data stored on local file systems, *OpenStack Swift*, *Amazon S3*, and *HDFS*. It offers an extensible query re-write mechanism that allows SQL to query data for users and external programs.

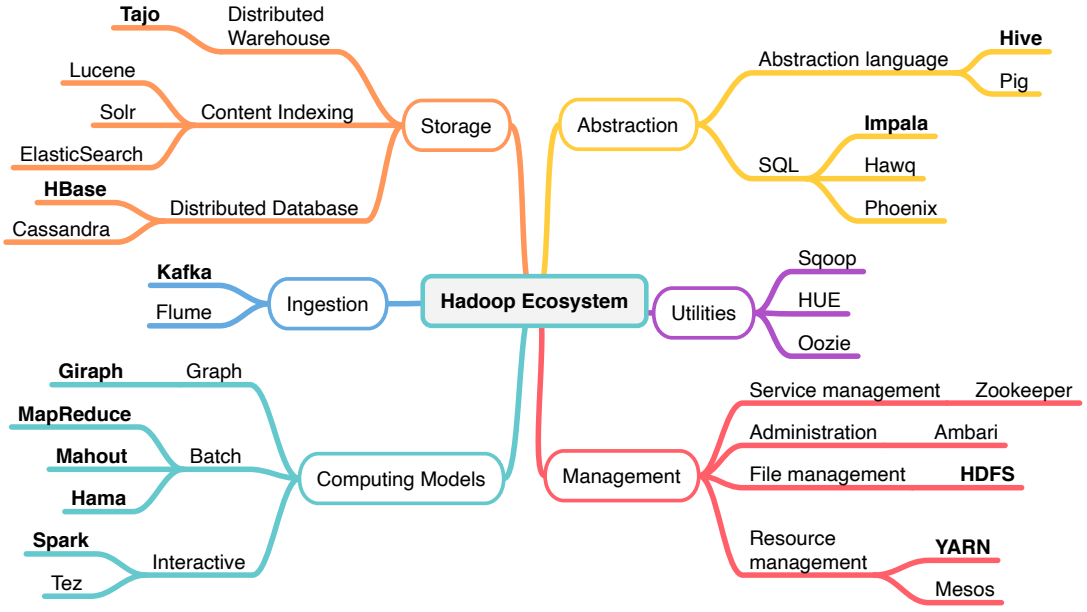


Figure 3.4: Mind map of the Hadoop Ecosystem

[1] http://www.impala.io
[2] https://hama.apache.org/
[3] http://mahout.apache.org/
[4] http://tajo.apache.org/

3.3/ SPARK

Hadoop architecture has pioneered the area of general-purpose data processing systems. However, one of the Hadoop system's key performance drawbacks is that before beginning the next one, it materializes the intermediate outcomes of each Map or Reduce stage on HDFS. For instance, the data transfer speed to an SSD equals to 600MB/s. However, the speed is ten times more (10 GB/s) when transferred to memory. Several systems have therefore been developed to resolve the Hadoop framework's performance issue. The Spark project has been implemented as a large data processing engine for general purposes that can be used for many kinds of data processing scenarios [40]. In theory, Spark was initially developed to provide interactive queries and iterative algorithms with significant efficiency, as these were two prominent use cases that were not well served by the MapReduce system. Spark, written in Scala[5] [41], was created as one of the new generation data flow engines designed to address the MapReduce system's limitations in the AMPLab at UC Berkeley and was open-sourced in 2010. In general, one of the key drawbacks of the Hadoop framework is that it needs to materialize the entire performance of each map and reduce the task into a local file on the Hadoop Distributed File System (HDFS) before the next level can consume it. This materialization process enables implementing a simple and elegant fault-tolerant mechanism for checkpoint/restart; however, it dramatically harms the system's performance.

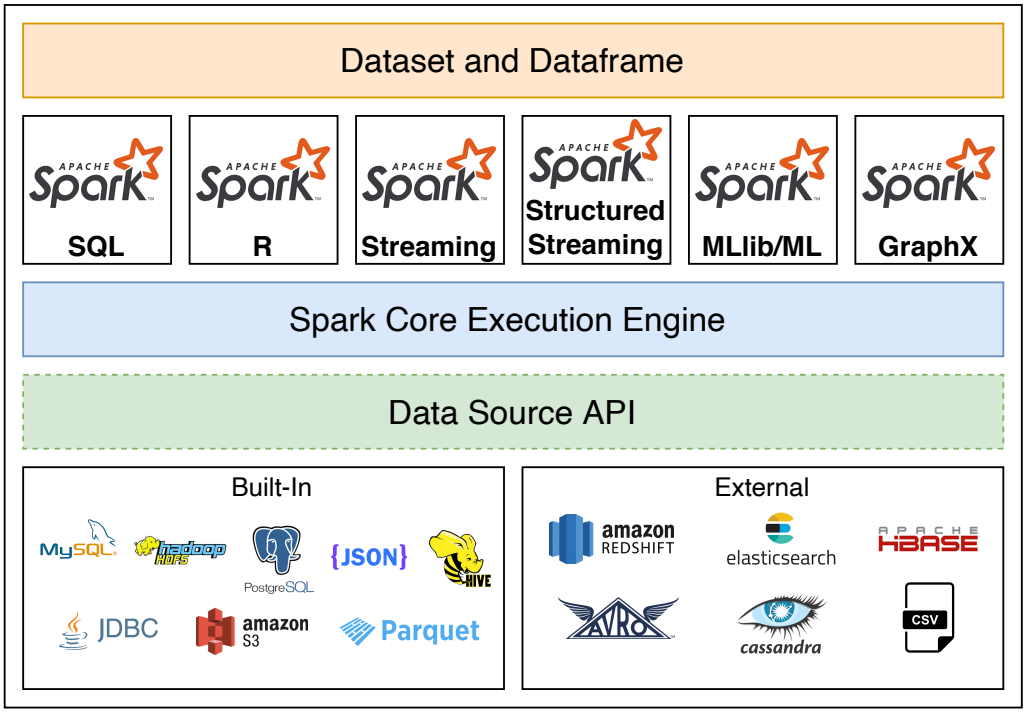


Figure 3.5: Ecosystem of Spark

[5] http://www.scala-lang.org/

By loading the data into distributed memory and depending on less costly shuffles during data processing, Spark brings Hadoop's ideas to the next stage. In particular, Spark's fundamental programming abstraction is called *Resilient Distributed Datasets* (RDD) [40], which describes a logical array of data partitioned across machines generated by referencing datasets in external storage systems, or by applying different and rich coarse-grain transformations (e.g., *mapping*, *filtering*, *reducing*, *joining*) to existing RDDs instead of the two basic *map* and *reduce*. For example, Spark's *map* transformation applies a transformation function to each element in the RDD input and returns a new RDD with the transformation output elements. Hence, the *filter* transformation applies a filtration predicate to the RDD elements and returns a new RDD with only the elements that fulfill the predicate conditions, while the *union* transformation returns a new RDD from the merge of the two input RDDs. a filtration predicate to the RDD elements. In theory, providing RDD as an in-memory data structure gives the power to the functional programming paradigm of Spark by allowing user programs to load and repeatedly query information into the memory of a cluster. Furthermore, in some MapReduce-like parallel operations, users can directly cache an RDD in memory across machines and reuse it. In particular, RDDs can be manipulated by operations such as mapping, filtering, and reducing, which take on programming language functions and send them to cluster nodes.

Spark can communicate with a broad range of distributed storage implementations like *Hadoop Distributed File System* (HDFS) [42], *Cassandra*, and *Amazon S3*. RDD may also be generated by distributing to existing RDDs a collection of objects (e.g., a *list* or *set*) loaded into memory or applying coarse-grained transformations (e.g., *map*, *join*, *reduce*, *filter*). Spark is highly based on functional programming principles. Functions reflect the basic programming unit in Spark, where functions can only provide input and output without state or side effects. Spark proposes two types of operations over RDDs in principle: *transformations* and *actions*. We use transformations to construct a new RDD from an existing one. One common transformation, for instance, is filtering data that matches a predicate. On the other hand, actions are used to evaluate a result based on an existing RDD and either return the results to the driver software or save them to an external storage device (e.g., HDFS). The RDD operations flow in Spark is shown in Figure 3.6. Other than the Spark Core API, additional libraries are par t of the Spark ecosystem with additional functionality in the Big Data processing area (see Figure 3.5). Table 3.1 provides an overview of some Spark's transformations, while Table 3.2 provides an overview of some Spark's actions. Spark offers numerous higher-level library packages in particular, including support for SQL queries [43], graph processing [44], streaming data, statistical programming, and machine learning [45].

Table 3.1: A non-exhaustive list of Spark's transformations

Transformation	Description
map	Apply a transformation function to each element in the input RDD and returns a new RDD with the elements of the transformation output as a result
filter	Apply a filtration predicate on the elements of an RDD and returns a new RDD with only the elements which satisfy the predicate conditions
distinct	Remove the duplicate elements of an RDD
union	Return all elements of two RDDs
cartesian	Return the Cartesian product of the elements of two RDDs
intersection	Return the elements which are contained in two RDDs
substract	Return the elements which are not contained in another RDD

Table 3.2: A non-exhaustive list of Spark's actions

Action	Description
take	Return number of elements from an RDD
takeOrdered	Return number of elements from an RDD based on defined order
top	Return the top number of elements from an RDD
count	Return the number of elements in an RDD
countByValue	Return the number of times each element occurs in an RDD
reduce	Combine the elements on an RDD together according to an aggregate function
foreach	Apply a function for each element in an RDD

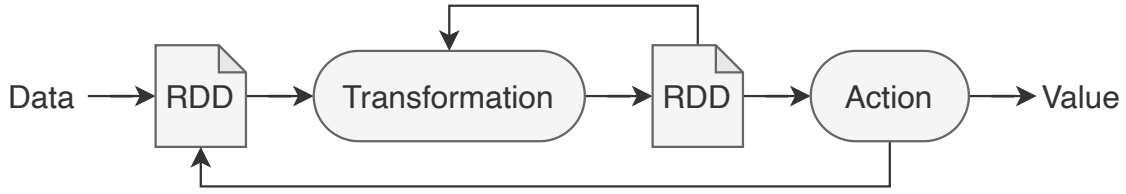


Figure 3.6: Flow of operations for RDD in Spark

3.4/ KAFKA AS A MESSAGING SYSTEM

Each enterprise is becoming a data-driven company. We collect data, evaluate it, exploit it, and generate more as a result. Whether it is log messages, user behavior, stats, machine sensors, or something else, every application creates data. Each byte of information has a story to tell, something of significance that tells the next thing to be done. We need to bring the information from where it is generated to where it can be processed to know what it is. On websites such as Netflix, where our clicks on films or series of interest are converted into recommendations revealed to us a little later, it is happening daily if not hourly. The earlier we can do this, the more our organizations can be agile

and receptive [46].

The less time we spend on moving knowledge around, the more we can concentrate on the core company at hand. That skill makes pipelining in the data-driven enterprise a critical component. How we transfer the data becomes almost as relevant as the information itself. Therefore, for those reasons, the Hadoop Ecosystem adopted this technology and became one main brick of it. In the following section, we briefly describe the messaging paradigm of Publish/Subscribe, then we give details about the Kafka technology used in our work to ensure a pseudo-exactly once processing of data.

3.4.1/ What is Kafka?

Apache Kafka is based on the *Publish/Subscribe* messaging pattern. It consists of two agents, the *sender* or *publisher* that sends a piece of data (*message*) with no particular destination and received by a receiver. The message sent by the publisher is tagged by identifying information, and then the *receiver* or *subscriber* subscribes to categories of tags. Pub/Sub systems typically have an intermediating point, usually called the *broker*. It is a middle point where published messages are stocked to be later on served for interested clients.

A./ Batches and Messages

Within Kafka, the unit of data is called a post. For database developers, Kafka may seem to be similar to a line or a record. As far as Kafka is concerned, a message is simply an array of bytes, so the data found within it does not have a particular format or significance to Kafka. There may be an optional bit of metadata for a post, referred to as a key. The key is just a byte array and has no particular significance to Kafka, as with the post. If messages are to be written to partitions in a more regulated way, keys are used. The most straightforward scheme of this kind is to produce a consistent key hash and then select the partition number for that message by selecting the total number of partitions in the topic as the hash module's product. This technique ensures that messages with the same key are written on the same partition at all times.

Messages are written in batches in Kafka for performance. A batch is just a series of messages, all of which are generated for the same subject and partition. For each message, an individual round trip through the network will result in unnecessary overhead, and this is minimized by grouping messages together into a batch. Batching is, of course, a tradeoff between latency and throughput: the larger the batches, the more messages per unit of time can be managed, yet the longer it takes to disperse each message. Usually, batches are also compressed, allowing more effective data transmission and storage

at some processing power expense.

B./ Topics and Partitions

Messages are grouped into topics in Kafka. A database table or a folder in a filesystem is the closest analogies to a topic. Additionally, topics are broken down into a variety of partitions. Back to the definition of ”commit file,” a partition is a single file. In an append-only mode, messages are written to it and are read from beginning to end in order. Notice that because a topic usually has several partitions, only within a single partition, there is no guarantee of message time-ordering across the entire topic. A topic with four partitions is shown in Figure 3.7, with writing appended to each one’s end. Partitions are also the way that Kafka gives redundancy and scalability. A partition can be hosted on a separate server, which means that it is possible to horizontally scale a single subject across multiple servers to provide output well beyond a single server’s capacity.

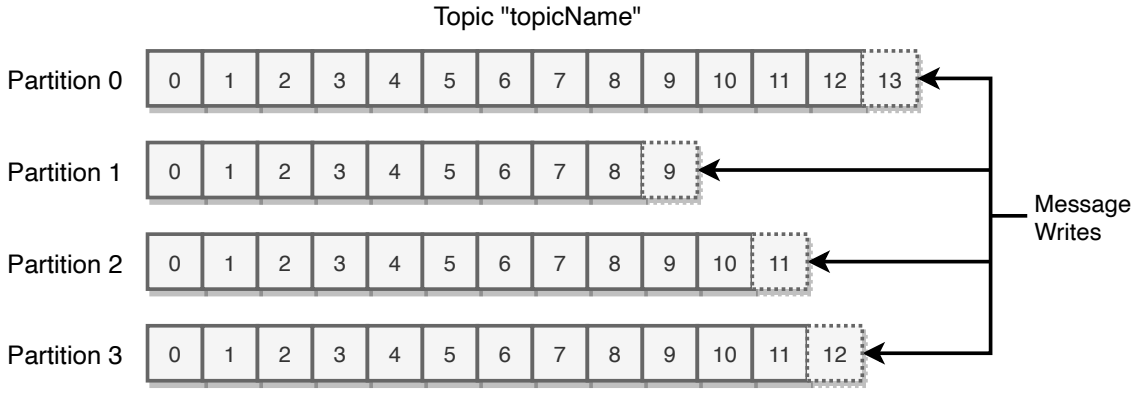


Figure 3.7: Representation of a topic with multiple partitions

When discussing data inside systems like Kafka, the word stream is sometimes used. Most often, regardless of the number of partitions, a stream is considered a single data subject. This atomicity reflects a single source of information that is transferred from producers to consumers. In discussing stream processing, this way of referring to messages is most common when frameworks, some of which are Storm, Kafka Streams, and Apache Samza, operate on real-time messages. This operation method can be compared to how offline systems, namely Hadoop, are structured to operate later on bulk data.

C./ Producers and Consumers

Kafka clients are machine users, and there are two specific types: producers and consumers. The Kafka Connect API for data integration and Kafka Streams for stream processing are both advanced client APIs. Specialized clients use producers and consumers

as building blocks and have higher-level features on top. Producers generate new messages. These may be named publishers or authors in other publish/subscribe schemes. In general, for a specific subject, a message will be produced. By default, the producer does not care what partition a particular message is written to and balances messages across all partitions of a topic. The producer can, in some instances, direct messages to unique partitions. Usually, this is achieved using the message key and a partitioner that produces a key hash and maps it to a particular partition. This mechanism ensures that all messages created with a defined key are written to the same partition. The producer may also use a custom partitioner that follows other business rules for mapping messages to partitions.

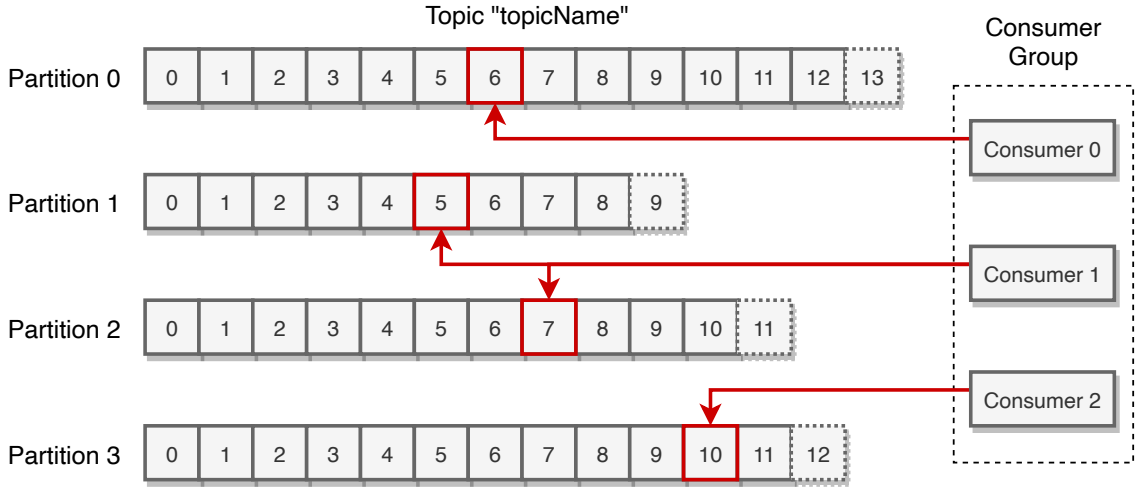


Figure 3.8: A consumer group (CG) reading from a topic

Consumers read messages. These clients may be named subscribers or readers in other publish/subscribe schemes. One or more topics are subscribed to by the user and read the messages in the order in which they were made. By keeping track of the offset of messages, the user retains track of whatever messages it has already consumed. The offset is another bit of metadata that Kafka adds to every message as it is made, an integer value that continually increases. There is a unique offset for any message in a given partition. A user can stop and restart without losing its position by storing the offset of the last consumed message for each partition, either in Zookeeper or in Kafka itself. Consumers operate as part of a group of consumers, one or more consumers working together to consume a topic. The group ensures that only one member uses each partition. There are three consumers in a single category consuming a subject in Figure 3.8. Two of the consumers work from one partition each, while the third client operates from two partitions each. Sometimes, the mapping of a consumer to a partition is called the consumer's possession of the partition. Consumers will, in this way, scale horizontally to consume subjects with a large number of messages. Additionally, the community's remaining members will rebalance the partitions being consumed to take

over for the absent member if a single user fails.

3.5/ CONCLUSION

MapReduce does not satisfy this interrogation because it uses HDFS, which means writing to files. Therefore, *MapReduce Online* [47] was created as a pipelining service to be run continuously, but between process, a pipeline's content was saved in HDFS. That is not suitable for streaming applications, where the data stream should be processed without relying on disk storage. Another solution was proposed to make the usage of HDFS more efficient. It consisted of adopting MapReduce for incremental computations that detect the input datasets' changes and enables the automatic update of the MapReduce jobs' outputs by utilizing a result reuse algorithm. That was made possible by using a modified version of the HDFS, the *IncHDFS* [48] (Incremental HDFS), that identifies similarities in the input data of consecutive job runs. In reality, Hadoop is built on top of the Hadoop Distributed File System (HDFS), a distributed file system designed to operate on commodity hardware, which is best suited for batch processing of vast volumes of data rather than interactive applications. This limitation makes the MapReduce paradigm unfit for event-based online Big Data processing architectures and motivates the need to explore other paradigms and novel frameworks for large-scale event-driven analytics solutions.

In the next chapter, we answer this question by detailing a new programming model tailored only for stream processing. This new paradigm allowed the construction of stream processing engines and platforms specialized in dealing with real-time data.

4

SPECIFIC-PURPOSE FRAMEWORK: BIG DATA STREAM PROCESSING

"παντα ρει (Panta Rhei),
*Everything **Flows**...*"

HERACLITUS

With the ever-growing flow of data originating from the Internet of Things (IoT) applications, whether it is smart cities, log monitoring, or business intelligence challenges, traditional processing paradigms attained their limits when it comes to storing or processing the data. In contrast to the previous chapter, this chapter focuses on a specific processing system, the stream processing platforms. In his book [49], Sakr highlighted how Hadoop is becoming an obsolete solution in the world of Big Data and that it could not be considered a "one-size-fits-all" solution since big data analytics application keep on growing in their variety and requirements. He coined a new term to describe a shift in processing engines' development, a new type of engines specific to domains and dedicated to particular axes. The term is Big Data 2.0, replacing the Hadoop traditional framework in different use cases. Figure 4.1 shows a timeline of the introduction of new processing systems. Since 2009, academic and industrial communities have focused on building the new generation of domain-specific big data analytics systems such as SQL processing platforms, graph processing platforms, and the focus of this chapter the stream processing platforms.

We then present an overview of what stream processing is as a technology, describes its history, and finally displays some examples of engines using that paradigm. We end by speaking about one of the main dimensions of an SPE: the resource allocation problem or query optimization, as a prerequisite to our second contribution developed in Chapter 6.

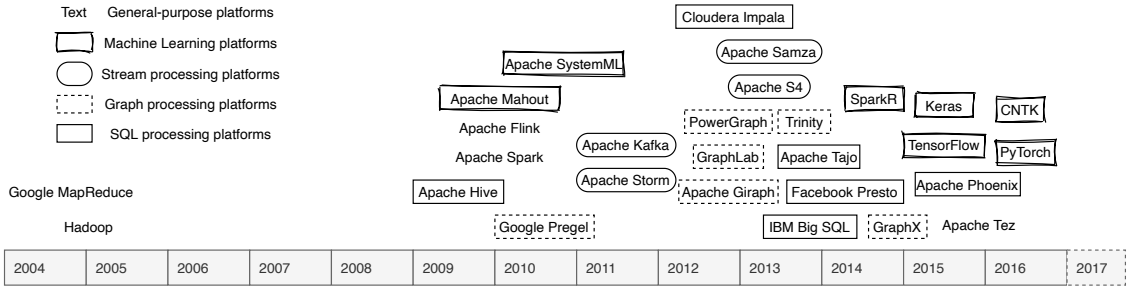


Figure 4.1: Timeline of Big Data processing platforms classified by processing type categories.

4.1/ INTRODUCTION

The availability of sensors, cell phones, and other devices has led to an explosion in the quantity, variety, and speed of generated data, which started to require some form of analysis. As society becomes more interconnected than ever, organizations produce vast amounts of data due to, among other reasons, instrumented business processes, user activity monitoring [50], wearable assistance [51], website tracking, sensors, finance, accounting, large-scale scientific experiments. Due to the challenges that it presents to existing infrastructure, such as data collection, storage, and processing, this data deluge is also called big data [52].

A large part of this big data, as it is generated, is most useful when analyzed quickly. Continuous data streams must be processed within very short delays in many evolving application scenarios, such as in the Internet of Things (IoT) [53], smart cities, and operational monitoring of large networks. In many contexts, data streams need to be analyzed to detect patterns, recognize faults [54], and gain insights.

Several stream processing systems and methods have been set up in a scalable and effective manner to perform analytical tasks and to answer the above requirements. Many systems implement a dataflow approach where incoming data items in data streams are redirected via a direct graph of operators placed on distributed hosts performing algebra-like or user-defined operations.

This chapter examines the stream processing computing paradigm along with the systems that implements such concept.

4.2/ ONLINE DATA PROCESSING ARCHITECTURE

Online[1] data analysis architecture is generally multi-tiered systems comprising of many loosely coupled components [55, 56, 57]. When it comes to structuring the architecture in such a way, the reasons are numerous. However, improving maintainability, scalability, and availability are always the primary goals of such architectures. Figure 4.2 provides an overview of the components often found in the architecture for stream processing.

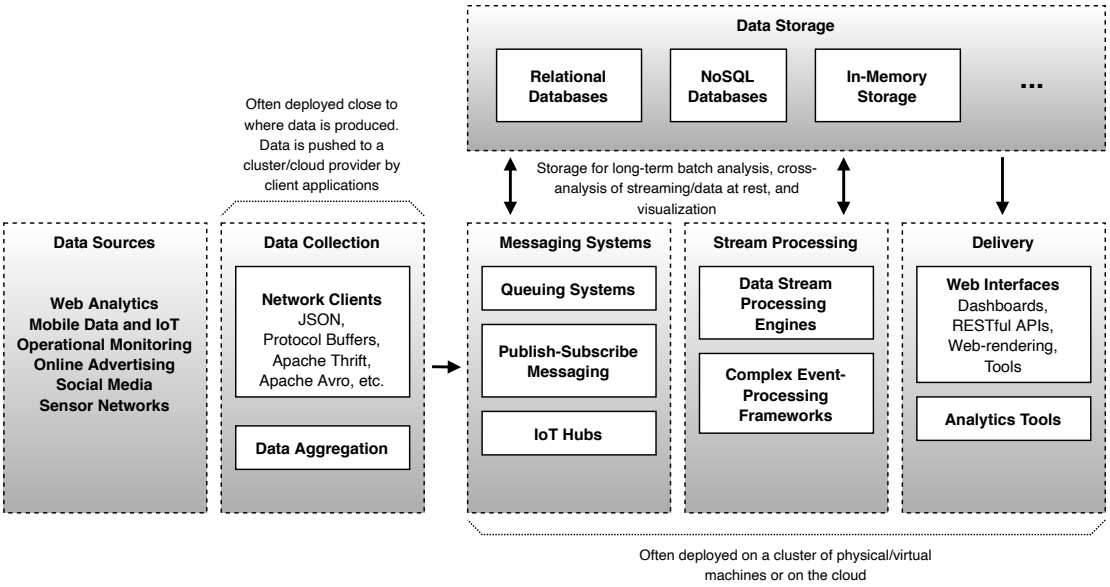


Figure 4.2: Overview of an online data-processing architecture. Taken from [58].

The **Data Sources** (Fig. 4.2) that demands real-time processing and analysis include Web analytics, operational infrastructure monitoring, online advertising, Internet of Things, and social media.

Most **Data Collection** is carried out by tools that run near the data location and communicate the data via TCP/IP or UDP connections, or long-range communication [59].

An online architecture for data processing may comprise multiple levels of collection and processing, with an ad-hoc connection made between those levels. To enable more modular systems and allow each tier to grow at different paces and thus accommodate changes, the connection is sometimes made through message brokers and queuing systems, such as Apache ActiveMQ (2016), Kestrel (2016), and RabbitMQ (2016). Other publish-subscribe based solutions, including DistributedLog (2016) and Apache Kafka (2016), or managed services such as Amazon Kinesis Firehose (2015) and Azure IoT Hub (2016). These systems are here referred to by **Messaging Systems**, allowing, for instance, the processing tier to extend to several data centers and collections that need to be modified without affecting processing.

[1] Hereafter use the term online to mean that "data are processed as they are being generated"

Several models and frameworks were created over the years for processing large volumes of data, among which MapReduce is one of the most popular [60]. Although most frameworks process data in a batch manner, there have been numerous attempts to adapt it to handle more interactive and dynamic workloads [61, 62]. Such solutions handle many of today's usage cases, but there is a growing need always to provide short response time services and to collect data on higher rates. **Data Stream Processing** systems are commonly designed to handle unbounded data streams and perform one-pass processing. This brick is the chapter's focus, and it includes solutions commonly known as stream management systems and complex event processing systems.

In addition, an architecture for data processing often stores data for further processing, or as support for presenting results to analysts or delivering them to other analytics instruments. There are various approaches for **Data Storage** to support real-time architecture, ranging from relational databases to key-value stores, in-memory databases, and NoSQL databases [63]. The data processing results (i.e., Delivery tier) are given to be used by analysts or machine learning and data mining software. Means of interfacing with these tools or displaying results to be visualized by analysts include RESTful or other browser-based APIs, Web interfaces, and other rendering solutions. There are also many cloud providers providing data storage solutions such as Amazon, Azure, Google, and others.

4.3/ DATA STREAM APPLICATION EXAMPLES

Streaming applications are programs that process data streams continuously. Due to increased automation in telecommunications, healthcare, transport, retail, science, security, emergency response, and finance, these applications have become omnipresent [64].

We address motivating scenarios in this section that inspire the need for Stream Processing Applications (SPAs). Examples can be found in domains ranging from financial markets, large-scale monitoring of infrastructure, manufacturing, traffic systems, healthcare, and safety, to climate and science in general. Check the list of applications' examples in Table 4.1.

We will exhibit two design scenarios in more detail in the rest of this section. Such examples cover transportation and healthcare applications. Each of them exhibits specific unique data processing and analytical characteristics [65].

Table 4.1: List of stream processing applications (SPAs)

Field	Application
Fraud Prevention	• Multi-party fraud detection • Real-time fraud prevention
Natural Systems	• Wildfire management • Water management
Stock Market	• Impact of weather on securities prices • Analyze market data at ultra-low latencies
Transportation	• Intelligent traffic management
Health and Life Sciences	• Neonatal ICU monitoring • Epidemic early warning system • Remote healthcare monitoring
Law Enforcement and Cyber Security	• Real-time multimodal surveillance • Situational awareness • Cyber security detection
Telephony	• CDR processing • Social analysis • Churn prediction • Geomapping

4.3.1/ TRANSPORTATION GRID MONITORING AND OPTIMIZATION

Application context Road networks and vehicles are becoming more instrumented with many different sensor types producing continuous data that can be analyzed by both offline applications and SPAs. For example, road networks are fitted with loop sensors that are installed under the pavement to measure traffic volumetrics, with sensors on toll booths, as well as cameras at various points of entry and exit, and intersections.
Similarly, most public transports, as well as commercial fleets, are now equipped with Global Positioning System (GPS) sensors that periodically report location and travel information [66].
Which of these sensors serves as a streaming data source with individual streams consisting of timestamped numerical and non-numeric measurement sequences, collecting data in various formats, and sampling at varying time intervals.
The stream processing applied to these data will allow numerous new applications, from real-time traffic monitoring to personal travel advisors and location-based services. Those scenarios are shown in Figure 4.3.
Real-time traffic grid monitoring includes constructing and updating a representation of their condition, including, for example, occupancy level and average speed of different connections in the road network. This aggregated information can be used to recognize unusual incidents, including congestion, injuries, and other issues with infrastructure. Besides, the availability of this type of live information will also allow public infrastructure agencies to make decisions that can alleviate traffic issues in some instances.

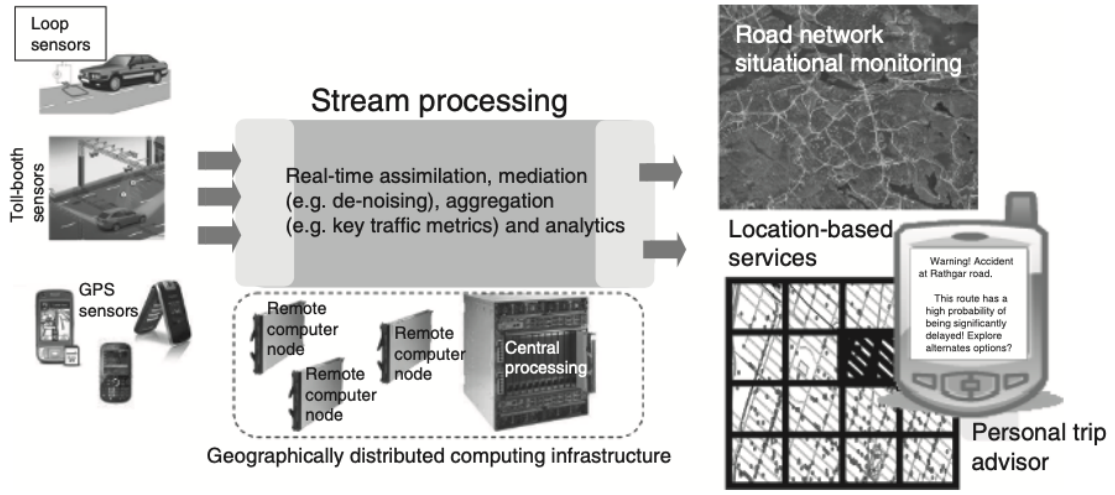


Figure 4.3: Stream processing in transportation. Taken from [65].

Application requirements In order to support these transport applications, the latency between when the data are generated and when action can be taken is within minutes. Latency is not as strict as in-network monitoring applications. However, near real-time processing is necessary.

In addition, multiple custom stream analytics are needed for various tasks ranging from parsing and extracting information from source data streams to data synchronization, accounting for missing values, de-noise, correlation, monitoring, and forecast.

Spatio-temporal analytics are also required to combine GPS and other position readings on maps, estimate travel direction, and distance, and calculate the shortest trajectories. The incorporation of these various types of information across vehicles, road segments, sensor types, and support for these complex tasks often involves the use of multimodal (i.e., representing different types of data) and multivariate (i.e., denoting multiple random variables) statistical analytics.

Although some of the individual low-complexity sensors (e.g., loop sensors, toll booth sensors) produce low data streaming data, approximately hundreds of bytes per second, a large road network may have several thousand or even millions of these sensors, resulting in high aggregate data levels. Besides, the streaming data often contains unstructured video and image signals with the use of multimodal information from cameras. Such systems also need to be flexible and adaptive.

Finally, the geographic distribution of the data sources and the use of inherently insecure network connections are a specific feature of applications in the transport domain. In these instances, using a distributed processing application model is also advantageous by putting computation near the sensors and hierarchically partitioning the analysis into local and central processing, as shown in Figure 4.3.

4.3.2/ HEALTHCARE AND PATIENT MONITORING

Application context Care for critically ill patients depends on the capture, analysis, aggregation, and interpretation of large volumes of data. The rapid and reliable incorporation of this data in the sense of a modern Intensive Care Unit (ICU) is essential for high-quality, evidence-based decision making. However, despite the increasing presence of new information technology (IT) devices in the ICU, many data analysis tasks are performed manually by most physicians, nurses, and other healthcare providers [67].
Initially, patient monitoring in the ICU was limited to clinical examinations and observations of a limited number of physiological readings, such as heart and respiratory rate, blood pressure, oxygen saturation, and body temperature. With advances in sensor technology, continuous monitoring of complex physiological data has become possible, enabling dynamic monitoring of changes in the patient's condition.
The data from these sensors provide more information to physicians than they can manually interpret, and it is not generally combined with other applicable information, such as laboratory test results and records of admission. As a result, physicians and other caregivers are often asked to integrate all of these disparate sets of information as best they can before they can determine the status of their patients and prescribe an appropriate course of treatment.

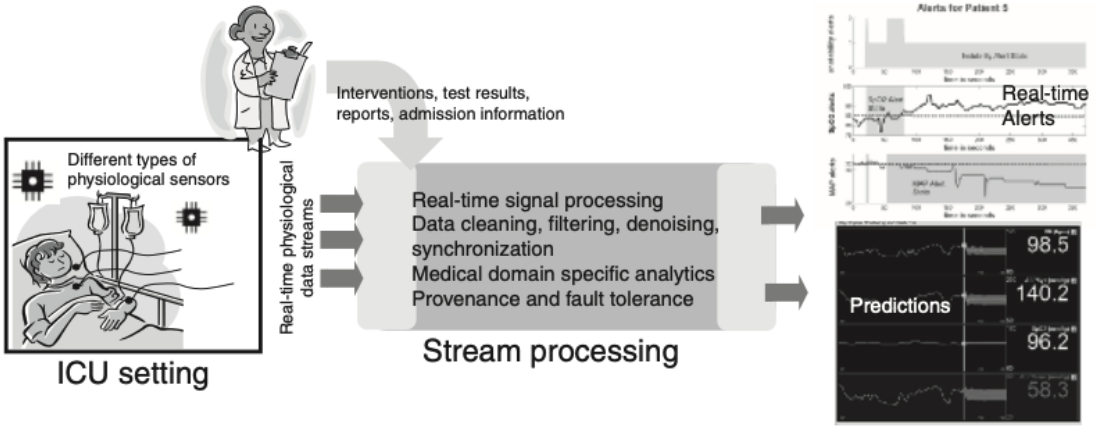


Figure 4.4: Stream processing in transportation. Taken from [65].

Application requirements Manual processing and management of healthcare data are fraught with many difficulties. First, data inflow can be both voluminous and too complex to process for humans without the resources of data analysis. Second, clearly periodic data reviews may add a considerable delay between a medically relevant incident occurring, and a caregiver's (possibly necessary) reaction. Third, manual analysis can miss essential subtleties in the data, mainly when the product of a combination of factors spread

through multiple data streams is an actionable event. As a consequence, medical treatment given at ICUs appears to be reactive in most cases. Often doctors respond to incidents that happened many hours or days ago, theoretically. Clearly, reducing reaction latency will lead to better patient outcomes. Finally, a typical ICU is an atmosphere of busy, complicated, and stressfulness. There is a limited margin for error associated with certain medical procedures. Often the workflow and the procedures may be both archaic and complex. Consequently, the amount of time that healthcare providers may devote to data analysis is often limited. By effect, the combination of all these variables results in not using the available knowledge from sophisticated instrumentation to its full capacity in real-time to assist in clinical management and in clinical research on a longer time scale. Therefore, the use of continuous applications and stream processing will improve healthcare delivery in many axes, including the automated collection of physiological data streams and other patient data in an ICU. Moreover, it will enhance its real-time use by electronic systems set up to detect and forecast medically critical events. These applications have a number of operational requirements.

The first criterion relates to the complex nature of the signals and the data which must be collected and processed continuously. Such data contains periodically sampled signals varying from tens to hundreds of kilobits per second (kbps) per patient, as well as intermittent events resulting from admission interview details, test results, and medical interventions. Hence, data ingestion through a continuous application includes cleaning, filtering, and data alignment operations, as well as multivariate statistical techniques to be developed.

The second requirement regards the design of medically relevant analytics, which combines the expertise of medical doctors (with domain knowledge) and computer scientists (with knowledge of continuous data analysis).

The third criterion concerns the introduction of frameworks for collecting and preserving audit trails of appropriate historical physiological data, along with features derived from the data, analytical results, and patient outcomes. It is crucial since patients also re-examine medical practices to improve health treatment.

The fourth criterion includes fault tolerance. Because of their direct effect on patient well-being and health outcomes, Healthcare SPAs must have uninterrupted uptime with no data loss.

The fifth criterion relates to data privacy. Data privacy must be covered in the clinical setting in order to avoid information from leakage and accessed by unauthorized parties. This necessity translates into a need for adequate protocols for authorization and authentication, data protection, and anonymization.

The final criterion corresponds to the collection and management of knowledge across extended time horizons across multiple patients. This requirement translates into the need for gracefully scaling up of the network and analytics to make use of an expanded

corpus of accumulated data.
Both of these criteria are expressed in the ICU scenario, as seen in Figure 2.4. As shown, multiple real-time physiological streams produced by patient-embedded sensors are combined with information from test results, treatments, and drugs to facilitate the ongoing monitoring and forecasting of their condition. A similar program, developed using the InfoSphere Streams SPE, was implemented in Toronto's SickKids Hospital [68].

4.3.3/ Discussion

Across these applications, the analytical requirements include the need for streaming capabilities in data intake and analysis, as well as the ability to continuously tune analytics in response to changes in data and knowledge that have been accumulated so far. From the system infrastructure perspective, the requirements include the capability to modify the physical structure of the application and adapt the processing to scale up and achieve high performance and fault tolerance while providing controlled and auditable access to the data being processed and the results of the analytical process.
In the following sections, we will discuss information flow technologies that shaped today's processing systems in making the above possible and describe how we can design and build analytics as well as production-grade applications to meet these requirements.

4.4/ Information Flow Processing Technologies

The requirement to process large volumes of data in real-time and generate continuous information with actionable intelligence has attracted considerable attention from the research community [69]. Several attempts have been made to develop information flow processing technologies that paved the way for current SPEs. It is possible to group the initial set of such information flow processing systems into four broad classes: active databases, Continuous Query (CQ) systems, publish-subscribe systems, and Complex Event Processing (CEP) systems. We explain these technologies in more detail in the rest of this section, in order to provide a historical background for the advent of stream processing technologies.

4.4.1/ Active databases

Active databases are used to support applications where data transformation and event detection tasks are continuous. They depend on the rules of Event-Condition-Action (ECA), which capture events, the conditions surrounding these events, and the actions to be activated when these conditions are fulfilled. ECA rules apply to data contained in

the database, and external sources of events are not permitted. Events from both within the database as well as from outside sources are permitted in open database systems. Open, active database systems are, therefore, generally more suited to the continuous processing of streaming data. Active databases were not designed for handling data rates associated with data-intensive large scale scenarios. If it comes to supporting user-defined operations, they have limitations. Active databases have greatly influenced the design of current events programming models.

4.4.2/ CONTINUOUS QUERIES

Continuous Queries (CQs) are used to communicate requests for tracking information. A CQ is a standing query that tracks updates of information. It returns results once the updates exceed the user-specified threshold. CQs have three major components: a query, a trigger, and a stop condition. An application also needs to implement adaptive mechanisms to cope with the update rate variations. Research and development behind CQ systems were heavily inspired by techniques employed by active databases as well as query processing on append-only databases. CQ systems rely on the regular evaluation of queries over ever-changing databases, relying on incremental query processing strategies to facilitate the re-evaluation of outstanding queries. SPEs take the operating mode one step further by continually moving the data through the queries.

4.4.3/ PUBLISH-SUBSCRIBE SYSTEMS

Pub-subscribe is a mechanism powered by events that decouple data producers from data consumers. Pub-sub systems can be classified into two categories: topic-based pub-sub and content-based systems. Each publication may have one or more topics associated with it in a topic-based pub-sub system. A subscription specifies one or three topics of interest for that publication. Most of the pub-sub content-based systems support atomic subscriptions, which are specified in individual publications. They support composite subscriptions, which are described using concepts such as sequences and repetition on a series of publications. CEP systems are based on the idea of complex events that are very similar to composite subscriptions in pub-sub systems. SPEs are similar to CEP systems in terms of their event-driven architecture, where a stream of alerts is produced when new messages are released.

4.4.4/ COMPLEX EVENT PROCESSING SYSTEMS

CEP systems have been developed as computing platforms where events can be recorded, analyzed, aggregated, combined, clustered, and dispatched to real-time an-

alytical applications. These systems usually relate precise semantics to the processing of data items based on a set of rules or patterns that articulate the tasks for event detection. Most CEP systems focus heavily on rule-based detection tasks, expressed as a function of event sequences, logic conditions, and models of uncertainty in terms of complex patterns defined. To sum up, CEP systems provide sophisticated support over data streams for the rule-based complex event detection. Most of these systems use a centralized runtime, reducing the scaling. They present solely limited assistance in the handling of unstructured data and complex analyses.

4.4.5/ ETL AND SCADA SYSTEMS

ETL systems are used to implement moderately simple data transformation tasks. SCADA systems support monitoring applications. ETL systems operate on offline stored data, producing output that will also be stored. While many SCADA platforms have limited support for event correlation, they tend not to provide in-depth analytics of the sort SPEs focus on and usually handle much lower rate data inflows.

4.5/ STREAM PROCESSING ENGINES

The earlier information flow systems may support some of the processing and analytical requirements associated with continuous data processing applications. Such systems are typically adequate to incorporate small-scale and precise streaming of data processing with low performance and tolerance criteria for faults.
SPAs typically incorporate optimized, adaptive, and evolving algorithms as part of their analytical role. Besides, these applications must be flexible, scalable, and fault-tolerant in order to accommodate greater and, in many cases, increasing workloads, with support for low latency or real-time processing.
Such specifications contributed to the development of the computational paradigm and SPEs for stream processing. While SPEs were developed by integrating several of the technology ideas that pre-dated them, they also required many advances to the state-of-the-art algorithms, analytics, and distributed computing infrastructure for processing.
We will discuss the stream processing paradigm in the remainder of this section, beginning with the type of data generally processed by SPAs, as well as introducing its basic concepts and constructs.

4.5.1/ DATA

Data source A streaming *data source* is a streaming data producer, usually broken down into a sequence of individual data items that can be consumed and analyzed by the application.

Data sink Similarly, a stream *data sink* is a consumer of results, also expressed in the form of a collection of individual data items generated by upstream applications.

Data model The following three concepts describe the stream processing *data model*'s characteristics.
The data model for stream processing can be traced back to data models that support database technologies, accurately the relational model. Nonetheless, two main differences exist to accommodate the richer set of specifications associated with continuous processing: first, the introduction of the *stream* as a primary abstraction and, second, the incorporation of a richer and extensible collection of data types, allowing greater versatility in what an application can manipulate. That said, the key components of this data model are briefly described below.
A *data tuple* is a fundamental, or atomic, data item that is embedded in a data stream and processed by an application. A tuple is similar to a database row because it has a set of named and typed attributes. Every instance of an attribute is associated with a value.
The *data schema* is the type specification for the tuple and its attributes. Consequently, the structure of the tuple is defined by its schema. A tuple attribute could be recursively defined by another schema, making it possible to create structurally complex tuples.
With this in mind, a *data stream* is a conceivably infinite sequence of tuples sharing a common schema, which is also a stream schema. Each tuple in a stream is typically affiliated with a specific time step (or timestamp) related to when the tuple was captured or generated, either in terms of the time of creation, time of arrival, time-to-live threshold, or, only, a sequence number.

Structured, semi-structured, and unstructured Streaming data can typically be divided into three broad classes: *structured*, *semi-structured*, and *unstructured*.
Structured streaming data includes data with, a priori, a known schema (or structure), and its data items are organized into triples of name-type-value. Semi-structured streaming data includes data with either nonexistent or not available complete schema (or structure). However, the data or some of its components are associated with self-describing tags that differentiate its semantic elements and enforce a hierarchy of records and attributes. Unstructured streaming data, on the other hand, consists of data in custom or

proprietary formats; in many cases, it has binary encoding and may include text, audio, video, or image data.
The data collected by an SPA is always a mixture of structured, semi-structured, and unstructured data, where the unstructured and semi-structured components are processed gradually, and their underlying structure is figured out step-by-step through the application's analytical operations.

4.5.2/ PROCESSING

When the data is available as tuples, the analytical components of the application will process it. In this section, we will outline the basic concepts and terminology of stream processing.

Operator The basic functional unit in an application is an operator. In the most general form, input tuples from incoming streams are applied to an operator by an arbitrary function, and output tuples to outgoing streams. However, it is also possible that when operating as a data source or sink, an operator can not process any incoming streams or produce any outgoing streams, respectively. An operator may execute many different tasks:

(a) *Edge adaptation*, a process consisting of translating data from external sources into tuples, which can be collected by downstream analytics carried out by other operators.

(b) *Aggregation*, a process involves collecting and summarizing a subset of tuples from one or more stream sources, bounded by various logical, physical, or temporal constraints.

(c) *Splitting*, a process that involves partitioning a stream into several streams for better use of data or task parallelism opportunities as well as for addressing specific application-specific needs.

(d) *Merging*, a process that involves combining multiple input streams with potentially different schemas based on the condition of matching or alignment, including temporal or other explicitly defined affinities.

(e) *Logical and mathematical* operations, tasks that involve applying various logical and relational processing, and mathematical functions to data items attributes.

(f) *Sequence manipulation*, a process that consists of reordering, delaying, or modifying the temporal properties of a stream.

(g) *Custom* data manipulations, tasks that involve the application of data mining, machine learning, or unstructured data processing techniques, potentially combined with further user-defined processing.

An *input port* is the logical communication interface that enables the reception of tuples. Likewise, an *output port* allows an operator to transmit tuples. An operator may have zero input ports or more, as well as zero output ports or more.
As discussed above, as a result of its internal analytical or data manipulation task, an operator generates typically new data. Generally, an internal analytical process has a long-lasting requirement for continuous processing, as opposed to a process-to-completion paradigm seen in traditional data processing projects. Besides, the analytical process is usually temporarily limited, either because specific tasks have to be performed in a given time or simply because the function's execution needs to be able to keep up with the incoming data flow.

Stream Streaming data in the form of a tuple sequence is carried through a *stream connection* between the upstream operator's output port and the downstream operator's input port. A stream connection can be physically implemented using a transport layer, providing a common interface that hides a low-level transport mechanism.

Stream Processing Application Typically, an operator works in conjunction with others by sharing tuples through streams. A set of operators and stream connections, organized in a data flow graph, are the components that define the *Stream Processing Application* (SPA).
An SPA is usually designed to perform a specific continuous analytical task. Thus, raw data flows into its source operators, which continuously convert them into tuples and send them for further processing by the rest of the operators in the application flow graph, until the sink operators finally send the results to external consumers.
The flow graph structure represents the *logical view* of the application, in contrast to its *physical view* (see Section 4.6). The flow graph can be arbitrarily complex in its structure with multiple branches, as well as possible feedback loops, which are typically used to provide control information back to the earlier stages of processing.

4.5.3/ SYSTEM ARCHITECTURE

The *middleware*[2] for ingesting, analyzing, and processing of streaming information is provided by a Stream Processing Engine (SPE). SPEs usually have two main elements:

[2]Middleware is the term used to denote the software which provides services beyond those available from the operating system to other software applications.

an application development environment and an application runtime environment.

Application development environment The application development environment sets out the platform and methods used for applications' implementation. This environment usually includes a (1) *programming model*, organized around a programming or query language to express the internal processing logic and structure of the application, and (2) a development environment in which the application can be interactively implemented, tested, debugged, visualized and finely tuned.
Typically speaking, the continuous essence of streaming data and analytics implemented by the SPA is best articulated by specific stream processing languages and programming models where abstractions exist for operators, ports, streams, and stream connections.
The programming model represented by a *stream processing language* provides guidelines for manipulating data as well as to implement, join subcomponents, and configure an application.
Mainly, a stream processing language includes constructs for (1) Defining a *type system* which captures the data structure manipulated by the application; (2) defining the application as a *flow graph* that translates the analytical requirements into a program that can be run; (3) implement data-manipulation mechanisms capturing the required buffering and subsequent operations to be performed; (4) Creating processing operators that encapsulate an analytical algorithm and allow application modularization and the reuse of common functionality across applications; and, finally, (5) Defining configuration and extensibility parameters for customizing an application for a specific runtime environment.
To support the implementation of SPAs, there are four separate types of programming languages: declarative, imperative, visual, and pattern-oriented.

Application runtime environment The runtime environment for the application includes the infrastructure that provides the resources and services to run one or more SPAs and manage their life cycle of execution. The environment consists of a software layer, theoretically spread through a multi-host environment where the applications are mounted along with the management elements of the system itself.

4.5.4/ IMPLEMENTATIONS

A./ APACHE STORM

Twitter presented the Storm[3] system as a distributed and fault-tolerant stream processing system that instantiates the basic principles of the Actor theory [70]. In Storm, the central

[3]https://storm.apache.org

abstraction is the stream. A stream is a series of unbounded tuples. Storm provides the primitives for the distributed and efficient transformation of a source to a new stream. *Spouts* and *Bolts* are the basic primitives Storm provide for performing stream transformations. A spout is a stream-source. A bolt absorbs any number of input streams, performs some processing, and can theoretically emit new streams. Complex stream transformations, including computing a stream of trending topics from a tweet stream, involve several steps and several bolts.

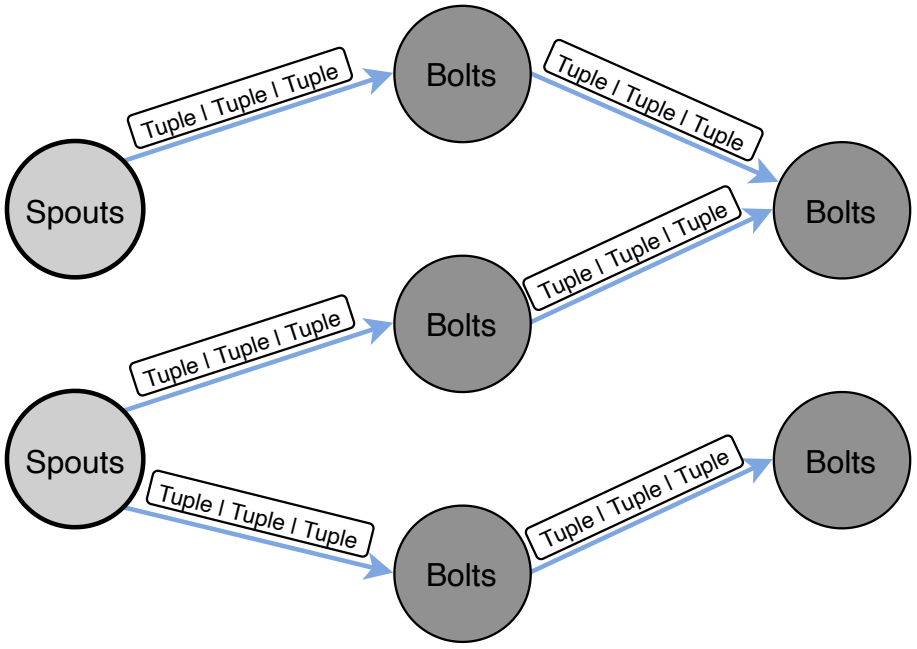


Figure 4.5: A Storm stream application topology

A topology is a transformation graph of a stream where each node is a spout or a bolt—edges in the graph shows which bolts subscribed to which streams. When a spout or bolt emits a tuple to a stream, it sends the tuple to each bolt that has subscribed to it. Links in topology between nodes show how the tuples should be passed around. Each node executes in parallel in a Storm topology. We may determine how much parallelism is required for each node in any topology, and then Storm will spawn that number of threads across the cluster to perform the execution. Figure 4.5 portrays a topology for the Storm study. The Storm system relies on the notion of stream grouping to determine how tuples are being sent between components that operate. In other words, it determines how to partition the stream among the bolt's tasks. Storm particularly supports various types of stream groupings, such as:

- *Shuffle grouping* where stream tuples are distributed randomly so that each bolt is guaranteed to have the same number of tuples.
- *Fields grouping* where tuples are partitioned with the fields defined in the grouping.

- *All grouping* where the tuples of the stream are copied and sent over all the bolts.
- *Global grouping* where a single bolt goes through the entire tube.

The Storm framework enables its users to establish their custom grouping mechanisms and the supported built-in stream grouping mechanisms. *Apache Trident*[4] was introduced as a Storm extension intended to be a high-level abstraction for real-time computing. Specifically, it plays the same role as high-level batch processing tools like Pig on the MapReduce ecosystem by offering built-in support for joins, aggregations, grouping, functions, and filters. Storm inspired the design of many other systems that followed, such as *Apache Heron*[5] and *Alibaba JStorm*[6].

B./ IBM STREAMS

IBM Streams[7] is a part of the IBM Big Data analytics platform that enables user-developed applications to ingest rapidly, analyze, and compare information as it comes from thousands of sources of the data stream. IBM Streams assesses a wide variety of streaming data — unstructured text, video, audio, geospatial, and sensor — helping companies recognize opportunities and risks and make real-time decisions. The system is designed to manage up to a million messages or events per second. It offers a programming model and IDE for identifying data sources, and software analytics modules called operators fused into execution units for processing. It also provides the infrastructure to support the composition from specific components of scalable stream processing applications. The key components of the Framework are:

- *Runtime environment*: This involves framework resources and a scheduler to deploy and track Streams applications through a single host or interconnected servers.
- *Programming model*: Streams programs can be written using a declarative language, the Streams Processing Language (SPL). Developers use the language to state what they want, and the runtime environment has to decide how best to service the request. A Streams application in this model is described as a graph consisting of operators and the streams which link them.
- *Administrative interfaces and monitoring tools*: streams data processing programs at speeds much more significant than those that standard operating system control utilities can manage effectively. IBM Streams offers the resources this environment can manage.

[4] https://storm.apache.org/documentation/Trident-API-Overview.html
[5] https://incubator.apache.org/clutch/heron.html
[6] https://github.com/alibaba/jstorm
[7] https://www.ibm.com/cloud/streaming-analytics

SPL, IBM Streams programming language, is a distributed language for the composition of data flows. It is an extensible, fully-featured language that supports user-defined types of data. A streaming application of IBM Streams defines a directed graph composed of interconnected elementary operators that operate on multiple data streams. Data streams may come from outside the system or be generated as part of an application internally. Fundamental building blocks for SPL programs are:

- **Stream**: A series of infinite ordered tuples. Operators can consume it on a tuple-by-tuple basis or by specifying a window.
- **Stream type**: Defines the name of each attribute in the tuple and the data type.
- **Operator**: SPL's fundamental building block, operators process stream data and can create new streams.
- **Window**: A finite group of sequential tuples. It may be based on the count, time, meaning of attributes, or punctuation marks.
- **Tuple**: A standardized list of attributes and the categories thereof. Any tuple on a stream has the form determined by their type of stream.

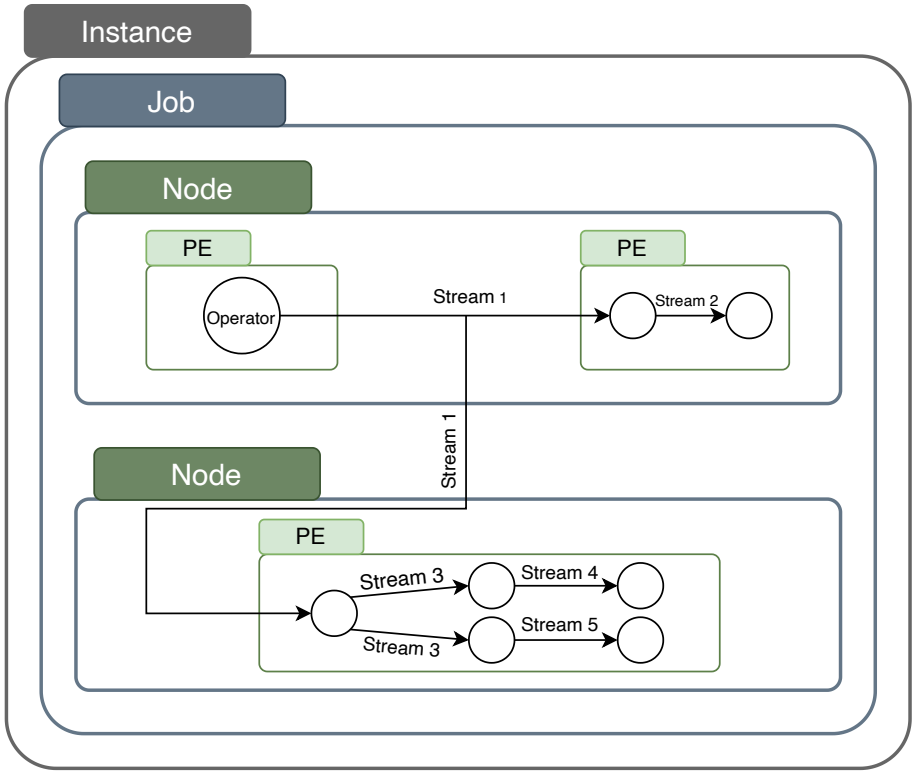


Figure 4.6: IBM Streams runtime execution

Figure 4.6 shows the runtime view of SPL programs on IBM Streams. An operator in this architecture represents a reusable stream transformer, which converts some in-

put streams into output streams. In SPL programs, operator invocation implements an operator's particular function, with particular input and output streams allocated, and parameters and logic specified locally. Every operator invocation names the streams for input and output.

4.6/ RESOURCE MANAGEMENT

Frameworks, which structure data stream processing applications as a data flow graph, usually use a logical abstraction to specify operators and how data flows between them; this abstraction is called a logical plan here (see Figure 4.7). A developer can present parallelization hints or specify how many instances of each operator should be instantiated when constructing the physical plan that is used by a scheduler or other component responsible for allocating resources from the available cluster resources to operator instances. As shown in the figure, the same logical operator 's physical instances can be placed on various physical or virtual resources.

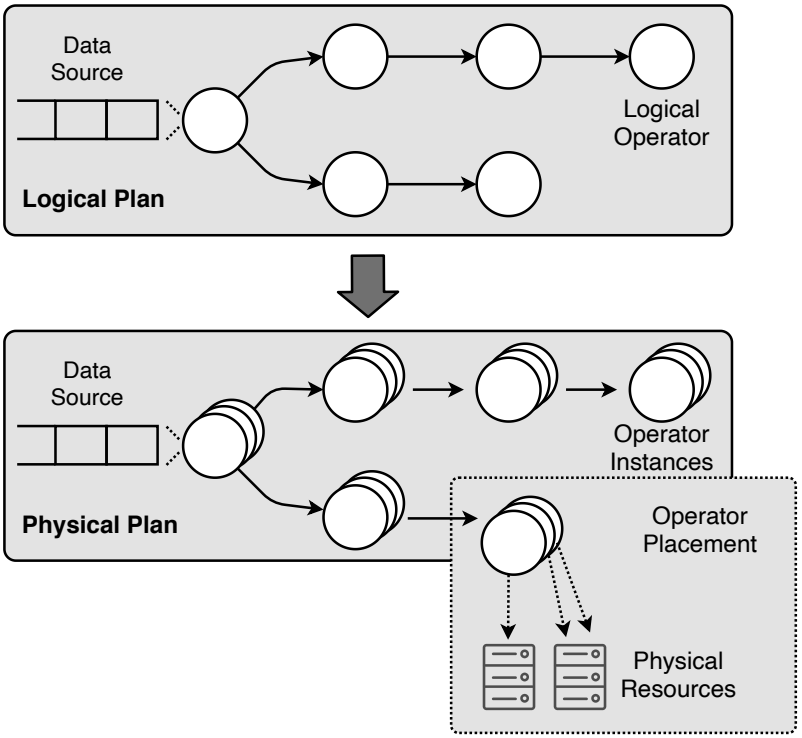


Figure 4.7: Logical and physical operator. Taken from [58].

Since operators run simultaneously, stream graphs expose parallelism inherently, but since many streaming applications require extreme performance, research communities have developed optimizations that go beyond that inherent parallelism. Digital signal processing, operating systems and networks, databases, and complex event processing are the communities that have focused most on streaming optimizations.

When we want to speak about *resource management* in the context of stream processing, we cannot move forward without speaking about an intrinsic characteristic of the latter, which is *distributed computing*. In the following section, we will review how SPEs are distributed and how applications (SPAs) leverage this architecture. More specifically, we will detail the logical and physical plans, how operators are placed onto resources, and finally, the transport layer inter-operator and inter-application.

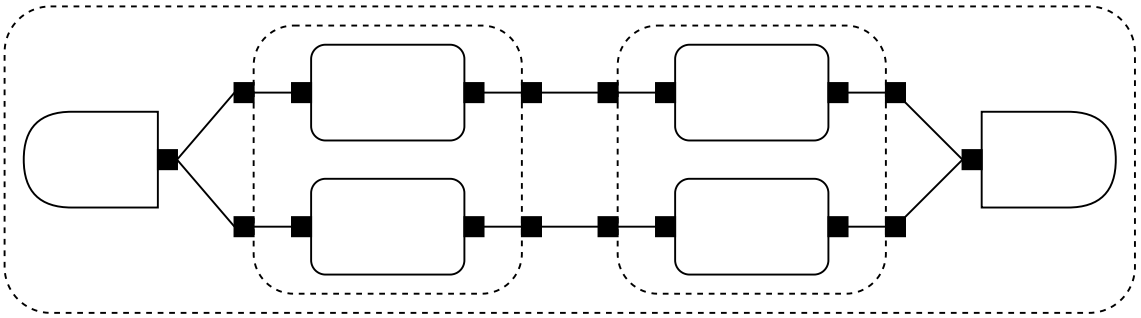
4.6.1/ DISTRIBUTED COMPUTING

A./ LOGICAL VERSUS PHYSICAL FLOW GRAPHS

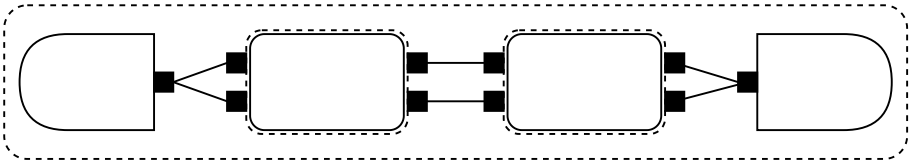
An application developer usually designs and implements an application as a *logical* data flow graph, where the network vertices correspond to the instances of the primitive and composite operator and the edges for stream connections.
Unlike this logical representation, when an SPE deploys an application, it will map it to a corresponding *physical* data flow graph, where the graph vertices correspond to OS processes and the edges to transport links that can transfer tuples between operators, possibly across hosts.
Figure 4.8a provides a logical description of an operation, including the full hierarchy of the operator instances. There are various viewpoints of varying granularities of the logical view, due to the presence of the composite operators. For example, Figure 4.8b shows the same function, but the instances of the composite operator collapse, and only the instance graph of the top-level operator is shown. A logical view always corresponds to the topology constructed by the application developer. However, it may present different perspectives of this topology with different levels of expansion of the composite operator. A significant observation is that a partition of the application will cross the boundaries created by a composite operator. The latter is merely a conceptual grouping motivated by such issues as encapsulation and reuse of code. The former, by comparison, is a physical grouping guided by issues like runtime efficiency and load balance.

B./ PLACEMENT

When an SPA covers several distributed hosts, the first decision a developer faces is how various parts of an application can be assigned to different hosts. The placement method takes two forms: *placement of partitions*, and *placement of host*.
Partition placement deals with OS *processes* associating to the operators. An application partition is a group of operators allocated to the same operation. Operator fusion is the mechanism by which they are put in the same process.
A *process* is a basic OS construct that represents the instance of a program being run.



(a) Logical application view, showing the complete operator instance hierarchy.



(b) Collapsed logical application view, showing the top-level operator instances.

Figure 4.8: Logical application topologies.

This offers separate address space and separation from other device operations, as well as its resource management (e.g., for memory, CPU cycles, and file descriptors) and context for specific execution threads. As a result, it becomes the basic execution unit in a distributed SPE when a partition is assigned to a process as each process can be mounted on a separate host.

Host placement deals with the association of partitions with the hosts which make up the runtime environment of the application. The method of assigning request partitions to hosts is called *scheduling*.

The combined process of creating application partitions along with scheduling brings a great deal of flexibility to the deployment of the SPA. Indeed, the same application will run on several different environments, from one single host to wide host clusters without rewriting it.

Although decisions can be delegated to the compiler (partition placement) and the application runtime environment (host placement) for all of these processes, an application designer can identify *constraints* and exert fine control over them if desired.

C./ TRANSPORT

The stream transport layer is used to implement physical connections that support logical stream connections used in the source code of the application. Flexibility and efficiency of the transport layer are essential for the development of scalable distributed SPAs.

Two aspects of the design of the transport layer are particularly important: first, the ability to be malleable to meet specific performance requirements, such as the focus on latency or throughput, and, secondly, the ability to use different types of physical connections to

deliver the most performance while imposing the least amount of overhead, regardless of the physical layout used by the application.
The performance requirements of the SPAs are in a continuum. However, it is important to look at the two extremes in this continuum: *latency-sensitive* and *throughput-sensitive* applications. Latency-sensitive applications usually have little tolerance for buffering and batching, as tuples must move between operators as quickly as possible, ensuring that fresh data is processed and that fresh results, including intermediate ones, are produced quickly.
Conversely, an application that is responsive to the throughput is involved in generating as many final results as possible in the shortest possible time. Thus, tuple buffering and batching techniques can give this type of application a significant boost in performance, especially in the early stages of its processing flow graph.

Selecting a physical transport Scalable SPEs have a range of different physical transports, tailored for specific situations and with different inherent costs and performance trade-offs. For example, stream connections between fused operators sharing the same application partition can be implemented very efficiently via function calls, since these operators are part of the address space of the same OS process.
Of course, the same is not true for connections between operators placed on the same host, but on different partitions for the application. In this case, one of several OS-supported Inter-Process Communication (IPC) abstractions can be used to enforce a physical link that corresponds to a stream: shared memory blocks, called pipes, or network protocols, like TCP/IP as well as Remote Direct Memory Access (RDMA) [71] over either InfiniBand or Ethernet.

4.6.2/ QUERY/PERFORMANCE OPTIMIZATION

Optimizing distributed applications typically requires innovation and an understanding of algorithms and system issues that could affect their performance. A variety of techniques and methods, including parallelization techniques that we discussed in the previous section, are part of the resources that developers can use. We address design concepts and implementation trends in this section that can assist in performance optimization tasks. They focus mainly on increasing the performance of an application, but we also address latency when it is adversely affected by the performance-driven optimizations. Performance optimization could be time-consuming and result in reconfiguring the data flow graph of an application, changing and improving the implementation of certain operators, selecting or tuning their analytics, as well as modifying how an application is deployed physically. However, it is crucial to building performance optimization skills in the context of stream processing, as many ongoing processing applications need to process a vast

amount of data promptly.

A./ Operator Reordering

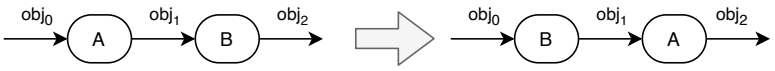


Figure 4.9: Adjust the order in which the operators in the graph appear.

The central principle of operator reordering is to hoist upstream selective operators so that certain data items can be discarded early. That way, by not processing these data objects, costly down-stream operators may spend less time. It is more profitable to put the less costly one first if operators A and B are similarly selective. It is more profitable to put the more selective one first if operators A and B are similarly costly.

B./ Redundancy Elimination

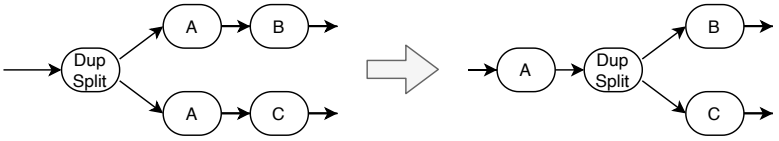


Figure 4.10: Eliminate the redundant operators in the graph.

Eliminating redundant operators, if resources are small, is profitable. For example, if a repetitive task takes time on a core that could be put to better use, it increases overall efficiency by removing that task. A com-mon redundancy trigger is a compilation based on simple templates being instantiated. Redundancy is not readily evident in some situations and often needs to be revealed to other optimizations. Multi-tenancy is another common cause for redundancy, where multiple users start related applications independently that can share subgraphs.

C./ Fusion

Figure 4.11: Merge several individual operators into one single operator.

Fusion is the dual separation of operators. Its key performance gain comes from decreased overhead contact and allowing conventional (non-streaming) compiler optimizations on the fused operator. However, fusion allows the same computer and theoretically

the same thread to be shared, while using fewer resources. In other words, there is less risk of mission or pipeline parallelism with fusion.

D./ FISSION

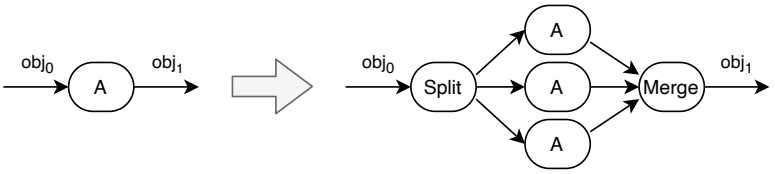


Figure 4.12: Replicate an operator for data-parallel execution.

Fission is beneficial when implemented on an operator with a high processing cost per data object when the parallelization overhead is minimal. In the ideal case, Fission can increase throughput on N cores by a factor of N. Speedups of 8 or even 16 are not unusual in practice, but the speedup is rarely optimal and ultimately usually caps out.

E./ LOAD BALANCING

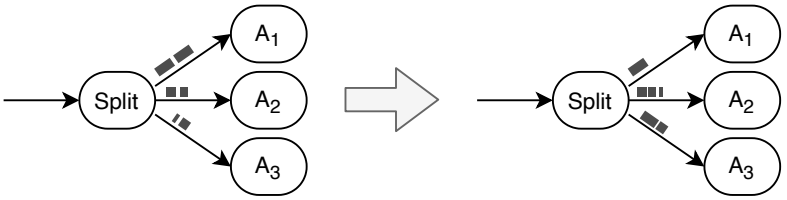


Figure 4.13: Adjust the order in which the operators in the graph appear.

The slowest operator typically bounds a stream graph's throughput. Load balancing aims to distribute the work equally so that all system nodes could run near capacity. Therefore, load balancing profitability depends on how imbalanced the load was to start with, and how well it can be balanced. In data parallelism (fission), load balancing can be achieved at the splitter by routing data objects to each operator replica, giving them about the same processing load.

4.7/ CONCLUSION

After exhibiting some of the known optimizations in the literature [72], in the next part of the thesis 'III. Contributions,' we dedicate a chapter on studying the fusion optimization and we propose a fusion strategy against the grain to popular belief after exploring the characteristics of a distributed application.

III

CONTRIBUTION

5

BIG DATA ARCHITECTURE OF GROUPE PSA

*"The beautiful thing about **learning** is nobody can take it away from you"*

B. B. KING

Nowadays, connected vehicles (CVs) are able to collect up to 170 different information (speed, temperature, fuel consumption, etc.) from onboard built-in sensors and transmit them, in a real-time fashion, to an infrastructure, usually by 4G/5G wireless communications. This reality raises many opportunities to develop new and innovative telematics services, including, among others, driver safety, customer experience, quality and reliability, location-based services, dealer services, infotainment, etc. 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 each day. Managing this big data, in real or batch mode, imposes tight constraints on the underlying data management platform. In this work, we report on real CVs big data platform, specifically, the one deployed by Groupe PSA[1]. In particular, we present technologies and open-source products used within different components of the platform to gather, store, process, and, more importantly, leverage big data and stress on why Hadoop system is not anymore the *de-facto* big data solution.

In this work, we describe the experience of PSA in building an automotive Big Data processing platform. The choices have made after comparing side by side other technologies that answered the same goal. The solution provided answered to Groupe PSA's requirements for a fast, reliable and ubiquitous platform for storage, cleaning and processing of automotive data. The research work in this contribution consisted of taking the existing architecture and re-invent it while improving its performance. In order to do

[1] Groupe PSA: a French car manufacturer. https://www.groupe-psa.com/en/

so, many proofs of concepts (PoC) were developed and tested. Subsequently, the PoCs were implemented to compose the current big data architecture studied in detail in this chapter and evaluated for its performance.

5.1/ INTRODUCTION

In Vehicle-to-Infrastructure (V2I) communication paradigm, Connected Vehicles (CVs), thanks to their various embedded sensors and telematics, can continuously feed the automotive infrastructure with a large amount of diverse useful data. CVs are equipped with a transmission unit (internal or external) that allows them to connect to a network (e.g., Internet) and communicate by sending and receiving data. For instance, currently, each connected vehicle of Groupe PSA can report over 170 different types of data (vehicle identification number, GPS coordinates, revolutions per minute, etc.) and PSA is expecting to handle millions of vehicles by the next decade [73] (Fig. 5.1 (a) and (b)). V2I, in contrast to V2V (Vehicle-to-Vehicle), offers many advantages that incite PSA to exploit this paradigm for its Connected Vehicle services. Some of its advantages are computation offloading, data persistence, network coverage, and accessibility.

A large amount of broad range data is a double-edged sword. On the one hand, it can be of great interest to automobile manufacturers, fleet managers, vehicle owners, and different other customers (Fig. 5.1 (c)). For example, automotive data can be leveraged to provide services to electric vehicles (such as battery pre-conditioning and charging), and it can also be leveraged to improve the driving experience, road safety, fleet management, and vehicle services. Table 5.1 lists a set of services that can be developed over the collected data and when consent and permissions are given by customers. On the other hand, before the automotive manufacturer can itself benefit from the collected data and can also offer information and services to its customers and partners, different challenges related to big data (such as storage, processing, analysis, leveraging, and sharing) must be tackled.

As regards innovative big data storage and processing, extensive research has been conducted on this matter, and numerous efficient solutions have been proposed to: (i) make up for the deficiency and inadequacy of traditional methods, and (ii) deal with huge data volumes [73, 74]. Presently, research is more focused on big data leveraging (such as advanced analytics and machine learning techniques on data lakes) [75, 76]. Despite all this work dedicated to big data challenges, to the extent of our knowledge, little attention has been devoted to the issue of proposing an end-to-end system; a complete model that is responsible for managing and protecting big automotive data flows throughout their life cycle [73, 74]: from the basic sensing operation to gathering, preprocessing, storage, processing, to the final leveraging and exploitation phase. An auto-

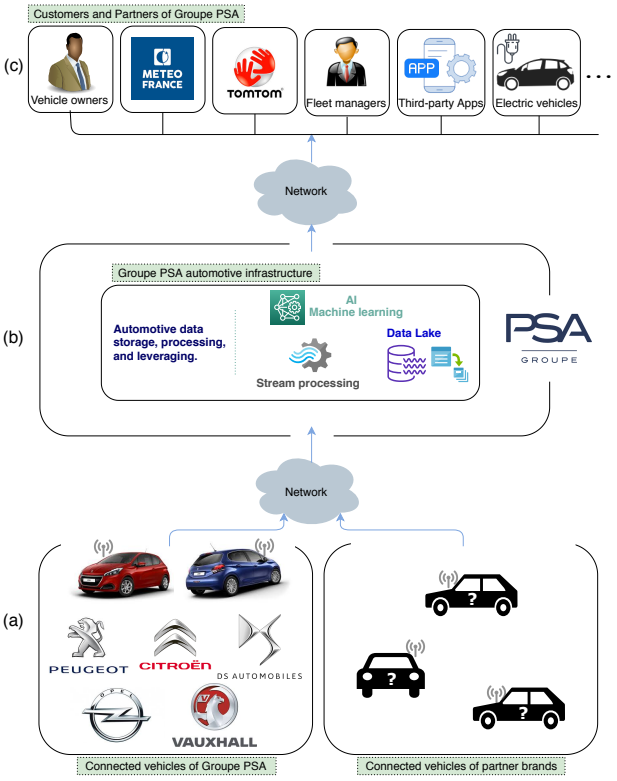


Figure 5.1: Big data gathering and leveraging by Groupe PSA.

motive big data gathering and leveraging architecture must adhere to the following four major requirements:

- **Time and space efficiency**: to provide smooth services to the final users, the proposed solution must use highly efficient storage and processing techniques. It must also support two mandatory processing modes; *online stream computing* and *data lakes*[1]. These two points will be detailed later on in this chapter.

- **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 inevitable increase of both.

- **Security**: first, the proposed solution must provide a customizable secure sharing. That is to say, vehicle owners must be able to share their data (with PSA partners, third-party applications, and so on) while being able to control what information can be accessed and by whom. Second, in addition to data privacy, all the necessary conventional security mechanisms must also be considered (confidentiality, integrity, and availability).

- **Robustness**: the proposed architecture must be robust and fault-tolerant. In case

[1] Data lakes were previously known as offline-batch storage and processing.

of any failure, recovery must be instantaneous, transparent, without data losses, and more importantly, without service interruption.

Cloud computing spans a range of classifications, types, and architecture models. The transformative networked computing model can be classified into three principal types: Public cloud, Private Cloud, and Hybrid cloud. Cloud computing can be offered in various models and deployment strategies, including the most popular Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The underlying architecture can take numerous forms and features, including virtualized, software-defined, and hyper-converged models, among others. Groupe PSA opted for the Hybrid cloud [77] for several reasons. First, because it offers the advantages of both the private and public cloud and yet remaining secure, scalable, and cost-efficient. One of the undeniable advantages of the hybrid cloud model is to have a private on-premise infrastructure that is directly accessible; in other words, it is not provided via the public internet. Access time and latency are, therefore, significantly reduced compared to public cloud services. Another advantage of a hybrid cloud model is the ability to have an on-premise computing infrastructure that can support the average workload for business while retaining the ability to leverage the public cloud for hot-recovery situations where the workload exceeds the computing power of the private cloud component. For instance, when it comes to Machine Learning/Deep Learning workloads that need a large but temporary processing capacity for training the models, Hybrid Cloud can provide an elastic overflow capacity and to have ”on-demand” processing capacity. Moreover, it reconciles the two major types of Cloud Computing, Big Data/Hadoop infrastructure, and the High-Performance Computing infrastructure. Hybrid cloud does that by connecting the computing farms to the storage warehouses. In such a setting, data locality becomes more flexible, and administrators would be able to execute or move data around to remain close to computing resources and therefore reducing the impact of the network latency. The impacts of adopting a hybrid cloud will be manifested in every component of the big data architecture detailed in the rest of this chapter.

To plug the previously discussed gap while satisfying the requirements mentioned above, the present work aims to report on a standard big automotive data model. To be more specific, the goal is to (1) provide a detailed description of the layered big data architecture of Groupe PSA and (2) succinctly narrate the state-of-the-art experience and feedback of this well-known automobile group in terms of gathering, storing, processing, and leveraging its continuous streams of vehicle-data.

The remaining of this chapter is organized into six sections, as follows. Section 5.2 gives some basic big data concepts, in particular, we re-discuss the 5 V's of big data in the light of automotive application. Section 5.3, the core of this chapter, describes the big data architecture deployed by Groupe PSA and details both its layers and fundamental

Table 5.1: Use Cases of the Connected Vehicle

Application	Description
Connected Services	- Last valid position. - Theft tracking. - Digital maintenance book of customer's vehicle. - Raising vehicle alerts. - Technical sheet of the vehicle thanks to the data of the connected vehicle and signature via the blockchain [78, 79]. - Eco-driving score + advice to reduce consumption.
Customized Vehicle Insurance	Pay-as-you-drive or the-way-you-drive
Smart Cities	Traffic and parking management through detection of free spaces
Weather	Temperature and humidity feedback in real-time
Added-value for OEM	- Improved product by analyzing the defects. - Adaptation of vehicle functionality in different countries (removal of unused functionality, for example, to lower costs).

technologies. Section 5.4 presents a few potential automotive applications and also gives some actual services that are currently provided by the PSA platform. Section 5.5 evaluated the proposed architecture by studying two dimensions: the performance and the quality of service and data. Finally, the last to sections concludes the work and suggests directions for future research and architecture enhancements.

5.2/ THE FIVE V'S OF BIG DATA

Although many definitions have been proposed to describe and explain the big data notion, no agreement on a single universal definition has been reached. In this work, the term ”big data” is used to refer to enormous datasets which due to their five intrinsic characteristics, known as the 5Vs, are challenging to handle:

- **Volume (data size)**: connected vehicles generate huge amounts of data at a very high pace. This data is sent to a dedicated automotive infrastructure to be stored and processed for immediate (stream processing) or subsequent (historical) use. Given the fact that traditional storing and processing solutions are inefficient in front of these new requirements and conditions [80], many novel highly efficient techniques have been proposed to solve this issue.

- **Velocity (data generation speed)**: among the basic intrinsic characteristics of automotive data is their very high growing tendency, which requires the use of the fastest and most efficient techniques proposed for big data storage and processing (stream computing, distributed file systems, distributed computing, ...).

- **Variety (data diversity)**: automotive data can be collected from numerous heterogeneous sources with various formats (it can be structured, semi-structured, or

unstructured). Thus, before being stored and processed, the collected data must be prepared and formatted to be usable.

- **Veracity**: the quality and trustworthiness of automotive data in terms of accuracy, completeness and consistency.

- **Value**: this last and most important V means that the collected big data is useless unless it is transformed into something meaningful and valuable. For instance, the collected data can help automotive manufacturers provide safer and better products/cars in the future. It can also help them provide services and applications to numerous customers and partners such as road safety, remote vehicle diagnostics, and electric-vehicle services. However, as previously stated, before vehicle manufacturers can take full advantage of the endless possibilities of automotive data and before they can provide all those interesting services, multiple challenges had, and still have, to be overcome. In other words, while some challenges related to big data have been efficiently addressed (such as data gathering, storage, processing, and sharing), others still need more investigations and efforts (e.g., advanced data analysis and big data leveraging using AI/machine learning approaches).

In addition to what has been said, to better meet the needs of customers and partners (through the different offered automotive services), two essential data processing modes must be considered: *online stream processing* and *batch processing*. These modes have two distinct purposes. The online stream processing processes raw data as it arrives (from connected vehicles) in a real or near real-time fashion, and is mainly related to connected B2C[2] services (such as fleet management and smart cities: traffic management, management of free parking spaces, etc.). As for the batch processing part, as its name might imply, it is dedicated to historical data processing, mainly, using efficient artificial intelligence and machine learning mechanisms. Moreover, automotive data (such as vehicles' identification numbers, GPS coordinates, and speed) cannot be shared and served to final users without being processed first. Indeed, vehicle-data must be processed according to the pre-established customers' preferences and consent. For instance, it can be averaged, filtered, or aggregated while considering a particular set of vehicles or a specific window of time. More details about all these points will be given in the next section.

[2]B2C: Business-to-Customer.

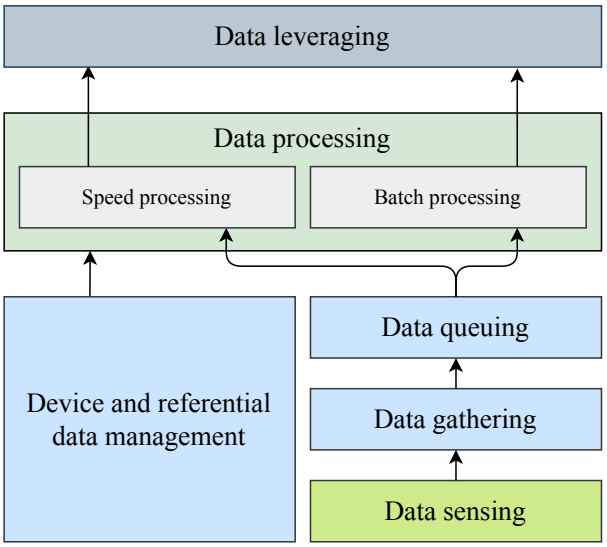


Figure 5.2: Layered big data managing architecture of Groupe PSA.

5.3/ MANAGING VEHICULAR BIG DATA: FROM GATHERING TO LEVERAGE

The present section details the main layers of the actual big data architecture that is currently deployed and utilized by Groupe PSA (Figure 5.2). We mention that the goal and policy of Groupe PSA focus on utilizing efficient open-source solutions whenever they are available and suitable for this hierarchical data leveraging structure.

As shown in Figure 5.2, the big data managing model of PSA consists of six layers: (1) data sensing, (2) data gathering, (3) data queuing, (4) device and referential data management, (5) data processing (with two different processing modes or sublayers), and finally (6) data leveraging and serving. Note that the first (data sensing) layer of this architecture is related to the connected vehicles of both Groupe PSA and its partner brands (Figure 5.1), while all the other remaining layers, from (2) to (6), are ensured by the PSA platform (MQTT broker, distributed streaming platform, centralized database, automotive infrastructure, data lake, etc.).

5.3.1/ DATA SENSING

The first basic layer in this big data architecture regards sensing and communicating ambient environmental information from connected vehicles to a dedicated automotive sink or base station (Figure 5.2). This essential step is mainly realized through the various embedded sensors and telematic service units (TSUs) of vehicles, which capture automotive data streams and report them via mobile wireless networks. The next paragraph will elaborate on TSU units and then address the protocol utilized to communicate data

from vehicles to the automotive sink.

Typically speaking, TSU units are installed in vehicles (connected to the CAN bus [81]) and have a data rate of 1 Mbit/s (according to the ISO 11898 standard [82]). Currently, Groupe PSA utilizes numerous kinds of TSUs, such as autonomous telematic boxes (ATBs), approved aftermarket boxes, and third-party telematic boxes. Each utilized TSU unit 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 TSU units, 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 fact, in addition to TSU units, V2X communications[3] can also be considered to send vehicles' data to smartphones and then to the automotive sink. In both cases (TSU or V2X), data is sent in a binary format using the well-known lightweight MQTT protocol. As shown in Figure 5.3, this communication paradigm connects, via a data broker, connected vehicles (publishers) with their automotive infrastructure (subscriber). In other terms, using their embedded sensors and telematics, vehicles gather the required data and send it, through the broker, to the automotive infrastructure.

Generally speaking, MQTT is based on a publish/subscribe messaging paradigm, where publishers publish about some specific topics (i.e., send messages to the MQTT broker). Subscribers (clients) can subscribe/unsubscribe to/from any topic proposed by the broker. The latter's job consists of forwarding the received messages from publishers to the appropriate interested subscribers (according to the recorded subscriptions). Finally, we mention that in MQTT, the publisher-broker and broker-subscriber communications can be specified using three different levels:

- QoS level 0 (at most once): transmitters send their messages only once without waiting for acknowledgments from receivers.
- QoS level 1 (at least once): in this case, acknowledgments are required for every sent message. That is, messages are delivered at least once (they can be sent/received multiple times).
- QoS level 2 (exactly once): guarantees that messages are delivered only once to recipients (a four-step handshake process).

5.3.2/ DATA GATHERING

The main role of this intermediate layer (called also front-end layer) is (1) gathering automotive data from the previous sensing layer and (2) preparing this input by making it

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

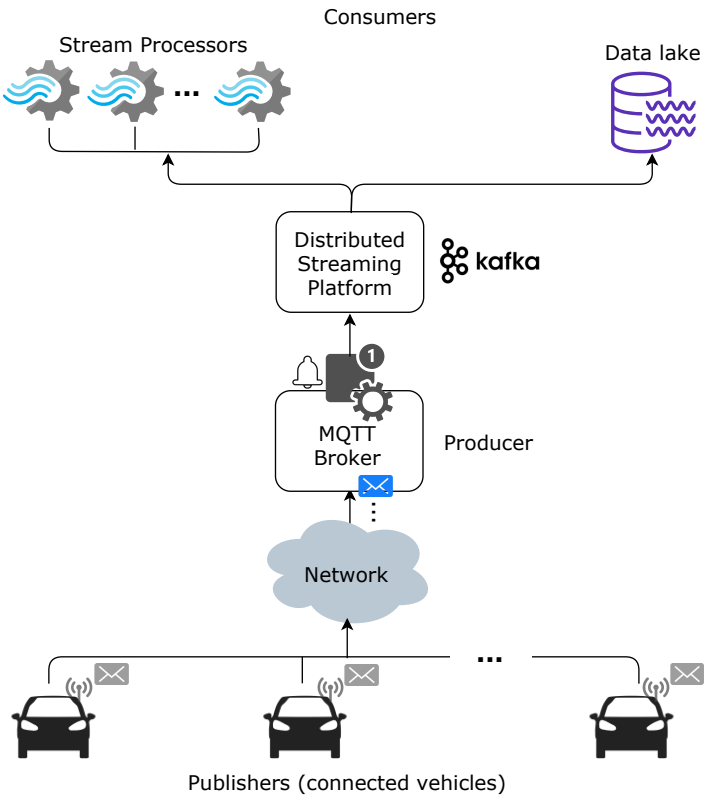


Figure 5.3: Big data sensing, gathering, queuing, and processing.

usable by the upper remaining layers. Regardless of the utilized communication protocol, the collected automotive data have to be ingested, decoded and formatted before it can be utilized. In the early versions of this big data architecture, the adopted communication protocol was HTTP/HTTPS and the main tasks of this layer which was an HTTP (Web) server were to receive the HTTP traffic, decode it, transform it into some specific format, and finally transfer the output to the upper (data queuing) layer. In the currently deployed architecture, as previously mentioned, the adopted protocol is MQTT. Accordingly, this front-end layer is an MQTT broker responsible for receiving, decoding, formatting, and handing the output to the upper layer (Figure 5.3). This choice of protocol is motivated by numerous reasons. MQTT, which is one of the most widely adopted IoT communication protocols, has been specifically tailored to be simple and easy to implement. Moreover, this application-layer protocol is convenient, more suitable, and has better performance than HTTP. Its lightweight and energy-efficient features offer light processing, fast communications, and allow devices with weak capacities to communicate with each other over low-bandwidth and unreliable networks.

5.3.3/ DATA QUEUING

From a theoretical point of view, this layer acts as an asynchronous secure producer-consumer queue. The aim is to cover any temporary unavailability of either the senders (MQTT Broker) or receivers (speed processing and data lake sublayers). This way, the real-time stream consumers and data lake will be resiliently and smoothly fed with the collected automotive data. The senders and receivers, in this system, do not interact directly, and messages are queued until being retrieved by the consumers.

The previously utilized message queuing (MQ) solutions (such as Message-Oriented Middleware software (MOM) and WebSphere MQ) suffered from several performance issues. The main one is that the adopted architecture was not fully distributed (it allowed the utilization of only one consumer, whether in the stream speed sublayer or data lake). To remedy this, currently, the popular and widely utilized Apache Kafka [83] is used to ensure the main tasks of this layer (Figure 5.3).

Before addressing the processing layer which relies in an indispensable manner on the three previously described layers (data sensing, gathering, 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. As shown in Figure 5.2, the three basic layers (data sensing, gathering, and queuing) and the referential layer are independent. The former ones are responsible for automotive data gathering and preparing while the latter is in charge of managing devices (activation, deactivation, update, etc.) and providing the necessary information required for the proper operation of the whole architecture.

5.3.4/ DEVICE AND REFERENTIAL DATA MANAGEMENT

In the currently deployed architecture, 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 layer. For instance, this referential layer is responsible for defining access policies, i.e., who[4] can access to which resource (environmental, geographical or other information). As a second example, the type of processing or even preprocessing, such as anonymization, filtering, and aggregation, that must be applied to the collected automotive data are dictated at this level.

Concretely speaking, as demonstrated in Figure 5.4, this layer, which is composed of a manager and a centralized database (that stores information mandatory for big data leveraging), serves as a reference for the upper processing and leveraging layers (re-

[4] Business-to-Business (B2B), Business-to-Customer (B2C), or both.

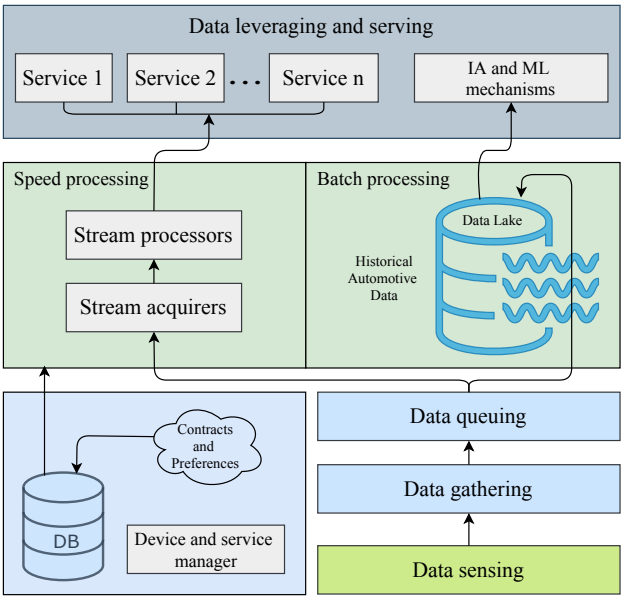


Figure 5.4: Detailed big data managing architecture of Groupe PSA.

sponsible for providing automotive services and applications to the final users). Vehicle owners, partners, and PSA administrators are granted access to this database and can interact with it through their respective spaces via the Web.

5.3.5/ DATA PROCESSING

This layer, which is considered to be the most important component of the whole PSA architecture, is divided into two fundamental data processing sublayers: data speed processing and data lake (Figure 5.4). While the speed sublayer is responsible for processing the continuous data streams (online processing), the data lake sublayer stores and processes historical automotive data with offline processing jobs (such as machine and deep learning techniques). As shown in Figure 5.4, the data lake sublayer can be pictured as follows. Automotive data flows continuously from the source (streams of queuing layer) to a dedicated lake. In other terms, this lake is directly fed by raw data coming from Kafka (Figure 5.3) and can be accessed and analyzed at any subsequent time. The next two subsections will elaborate on these two respective sublayers.

A./ DATA SPEED PROCESSING

This sublayer is responsible for two main tasks: data stream acquisition and data stream processing. As Figure 5.4 demonstrates, these tasks are ensured by two communicating streaming applications (or microservices) [84]:

- *Stream acquirers*: the role of these microservices is retrieving fresh automotive data

from the lower queuing layer and delivering it to the stream processors.

- *Stream processors*: these applications, which receive automotive data from the acquirers and retrieves referential data from the reference layer (Figure 5.4), has one fundamental task; processing the continuous data flow with strict constraints (i.e., online real or near real-time computing) while taking into account vehicle owners' preferences. As previously stated, the received streaming information cannot be shared as it is for the end-users. An aggregation function on a particular set of vehicles or an average on a specific window of time must be performed before sharing.

The remainder of this subsection addresses two basic concepts: stream processing and streaming applications.

A) Stream processing

Stream computing is a new programming paradigm that is designed for distributed and parallel processing of unbounded data streams. This paradigm is often confused with real-time processing, which requires a response within a certain period. To better explain the stream processing, the following points describe some of its main characteristics:

- Data items are processed as they arrive (i.e., online).
- Events are time-based. Typically, every record is timestamped on creation.
- Operations are done in a data flow fashion/design.
- Every operation is done on one data element (or a small window of the recent data).
- An operation calculates something relatively simple.
- Each computation needs to be complete in (near) real-time to avoid congestions.

To guarantee that the input data will be processed as it arrives, all operators must have a processing rate that is greater or at least equal to the input rate. For instance, if data generation frequency is 100.000 events/s, but there exists one operator that can process only 90.000 events/s, this will cause congestion that can go up to the source. To satisfy such hard constraints, stream computing considers various levels of parallelism (i.e., pipeline, task, and data parallelism).

B) Streaming applications

Streaming applications are programs responsible for processing continuous data streams. In more concrete words, a streaming application is composed of a set of operators that are interconnected via stream connections (Figure 5.5). As shown in this figure,

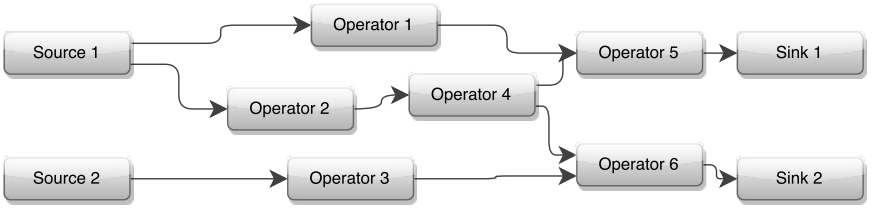


Figure 5.5: Streaming application example.

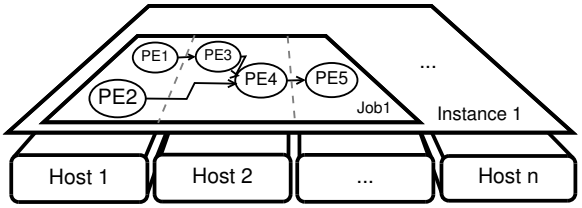


Figure 5.6: Illustration of the stream runtime environment.

a streaming application can be modeled as a directed graph where vertices represent operators and edges are data streams (continuous series of tuples generated by operators). Note also that in a streaming application there are three types of operators:

- *Source operators*: represent the source of input data streams. These operators are connected to an external source of data and can adapt the input data stream before sending it out through their output ports.

- *Computing operators*: carry out the necessary computation. They transform input data streams to the required output data streams by applying some operations (e.g., aggregation, correlation, etc.).

- *Sink operators*: having only input ports, these operators are usually placed at the end of the streaming application.

As depicted in Figure 5.6, streaming applications are executed as jobs on a *Stream Runtime Environment* (SRE), known also as *Instance*. Note that a job can be deployed on single or multiple processing nodes. Note also that a job is a running application that is executed as a set of Processing Elements (PEs), which execute each a set of operators. Given that jobs can be executed on multiple hosts, PEs within a job can communicate using Inter-Process Communication (IPC). As examples of streaming engines, we mention IBM STREAMS [85] (which uses TCP as IPC) and Apache Samza [86] (which uses Apache Kafka as IPC).

B./ DATA BATCH PROCESSING

This sublayer has a twofold role: (a) providing efficient distributed *storage* for the automotive data, and (b) providing fast, efficient offline *processing* for the historical data. First and foremost, it is worth mentioning that the collected automotive data is personal and sensitive and cannot be utilized in its arrival raw state; it must be pre-processed (anonymized and filtered) before use. To do so, the French authority responsible for data protection (CNIL)[5] has set strict laws and regulations. One of the latest European Union (EU) regulations on data protection and privacy is the General Data Protection Regulation (GDPR)[6], and every business that has a digital presence online has to comply with those rules. Groupe PSA is no exception to this rule; it anonymizes and filters automotive data according to the customers' preferences and consents. Each customer has a contract that lists all his preferences (e.g., anonymization, partners that can access data, etc.). These contracts can be accessed and modified by customers at any time via the Web.

Now, to be able to efficiently store the huge volumes of data generated by connected vehicles, distributed file systems with the following features are a must:

- *Efficiency*: the utilized system must provide efficient fast managing of large data volumes (such as fast read/write operations, etc.).
- *Scalability*: the considered distributed system must be able to support the huge and continuously increasing volumes of automotive data both in storage and processing (e.g., it allows adding more nodes/machines to the cluster).
- *Robustness*: the proposed system must be highly robust and available (for example, through data striping and replication across multiple cluster-nodes).

Before moving to the last layer in the architecture (i.e., the leveraging and serving layer), the following two subsections will talk more about automotive data storage and offline processing. These two subsections will also address the products that Groupe PSA has abandoned, and the new ones that have been considered to ensure efficient storage and processing for historical automotive data.

A) Data Lake

Pentaho CTO James D. introduced the concept of Data Lake back in 2010. As he has mentioned in his blog post, « *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*

[5]National Commission for Computing and Freedom (in French; "Commission Nationale de l'Informatique et des Libertés").

[6]Official legal text can be found here: https://gdpr-info.eu/

source to fill the lake and various users of the lake can come to examine, dive in, or take samples » [87].

A ”Data Lake” is a technique facilitated by a vast data repository based on low-cost technologies that enhance the collection, refinement, archival, and mining of raw data within an enterprise. 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 concept is simple: Instead of putting data into a data store designed for specific purposes, transfer it in its original format into a data lake. This technique reduces the upfront costs of data ingestion, such as transformation. Once data is deposited into the lake, it is available for analysis by all teams of the organization. The primary reason for implementing Data Lakes is the need for improved agility and accessibility for data analysis [88]. Some of the data lake's capabilities are listed below:

- To capture and store raw data at scale for a low cost.
- To perform transformations on the data.
- To store many types of data in the same repository.
- To define the structure of the data when it is used, referred to as schema-on-read.
- To perform new types of data processing.
- To perform single subject analytics based on particular use cases.

The data lake comes as an improvement to what data warehouses were limited to do. Data warehouses have been for some time the *ipso facto* solution for enterprises when it comes to storing data for later usage. As the big data wave arrives, however, businesses that have invested plenty of money constructing enterprise data warehouses (EDW) begin building data lakes.

The enterprise data warehouse (EDW) was developed at most organizations to collect information from several different sources so that reporting and analytics could be of use to everyone. The data warehouse for businesses was designed to create a single version of the reality that could be used recurrently. The data warehouse is a highly designed system. It often has a very complicated data model, which is carefully designed before loading the data.

Furthermore, the data warehouse supports the batch workloads and is designed for continuous use by hundreds to thousands of concurrent users who conducted reporting or analytics tasks. The answer is very straightforward, a hybrid and unified system includes the data lake and the enterprise data warehouse, where users can ask questions that can be answered by more data and more analytics with less effort.

At first, PSA adopted the Hadoop distributed file system by Apache, also known as HDFS (Hadoop Distributed File System) [89]. In a nutshell, this system has a master-slave cluster architecture that is composed of one *nameNode* and multiple *dataNodes*. While the master manages the cluster and stores meta-data files, slave nodes are responsible for data (or block) storage. In other terms, to be stored in this distributed system, files are split into blocks and then spread on the different *dataNodes*. Despite its distributed nature and high availability, HDFS suffers from some limits:

- Scalability, in HDFS, does not allow scaling up the storage without the processing. This limitation is a big issue in a hybrid cloud environment.
- The *nameNode* machine represents a single point of failure (SPOF) in the HDFS cluster. When it fails, the whole system will crash, and manual interventions become indispensable [89].

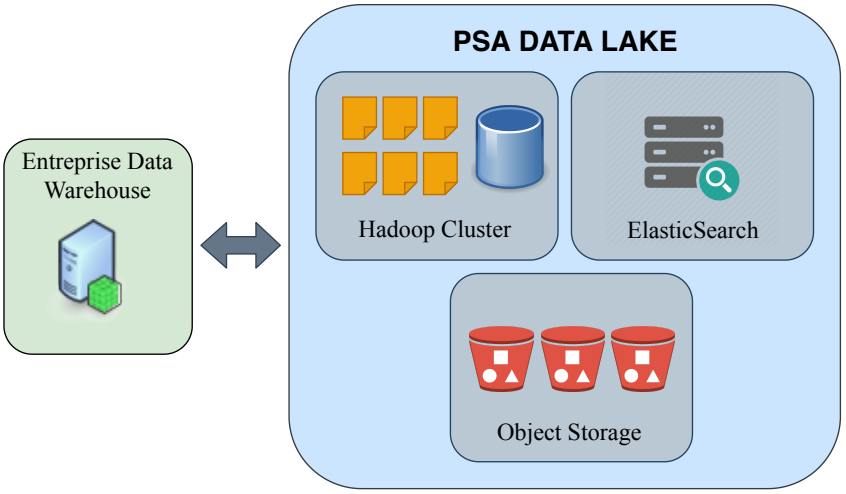


Figure 5.7: Illustration of PSA Data Lake

To overcome the numerous drawbacks and limitations of HDFS, Groupe PSA has opted for GPFS; an IBM distributed file system. Besides its full POSIX compliance feature, GPFS has several other advantages. For instance, unlike HDFS, GPFS has no single point of failure. This drawback is overcome by distributing both data and meta-data across the cluster disks (also known as *network shared disks* or NSDs). At the same time, Hadoop improved its file system by introducing for its NameNodes, the High Availability feature. It enables administrators to run redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. This feature eliminates the NameNode as a potential single point of failure (SPOF) in an HDFS cluster. As of Hadoop 3.0, admins can configure more than one backup NameNode.

Concurrently, PSA invested in deploying an object storage solution [90] in the goal to decouple the compute from the storage. Object Storage is an architecture for computer

data storage that manages data as objects, an alternative to file storage that manages data as a hierarchy of files, and blocks storage that manages data as blocks within sectors and tracks. Object storage composed of extended metadata. The unique identifier attributed to each object lets servers to retrieve it from any physical location. Use cases of object storage include cloud storage, photos, video, audio, and large image files. Object Storage can address the current Hadoop challenges in many ways. The key challenges are:

- **Accessibility and Durability:** *nameNode* could still be single point for failure in Hadoop. If *nameNode* fails, it would be difficult to access the rest of the cluster. With object storage, data accessibility or data loss are prevented and secured through a data protection mechanism known as erasure coding. Alternatively, if one instance of Hadoop fails, the data can be made available on the other instance.
- **Elasticity:** Object storage works on a pay-as-you-go consumption model. Users are only charged with what they are using at a certain time. Data can be added anytime it is needed. Transparently, the cloud provider automatically provisions resources on demand. Object storage is elastic, HDFS is not.
- **Scalability:** HDFS does not allow independent scaling. In Hadoop, compute power and storage capacity are tightly coupled, meaning that when a resource is added, the other must follow. On the other hand, object storage can easily scale out beyond Petabytes. Data can be easily accessed and processed on Hadoop when it is stored on object storage.
- **Cost:** A direct impact of the previously discussed point is the reduced cost of storage when storage and computing are separated. Not only does this separation lower costs, but it also increases efficiency. The cost of object storage is around the fifth that of the Hadoop platform.

Figure 5.7 illustrates the components of the Groupe PSA Data Lake and its interaction with the EDW. PSA Data Lake is composed of two main components, the less adopted solution, the Hadoop system, and the current go-to solution, the Object Storage. In parallel, it encapsulates an Elasticsearch engine. Essentially, Elasticsearch is a tool for indexing data, not exclusively for storing data. Elasticsearch may be incorporated in a data lake for indexing the data stored in it. Better again, if an organization 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 find data across multiple storage systems quickly. By doing so, moving data around will become simpler. For instance, to copy data to HDFS for a MapReduce job, or read directly from the source to load in memory to run in Spark.

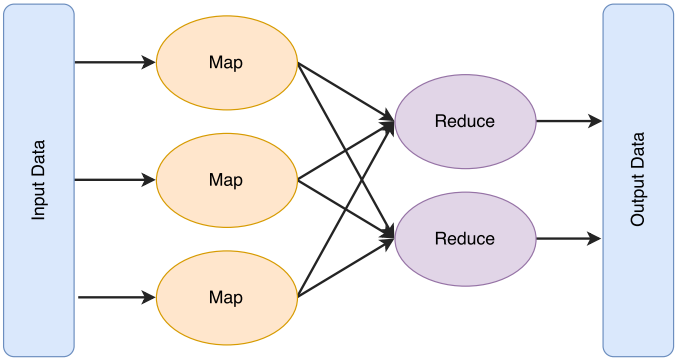


Figure 5.8: Example of a MapReduce job.

In conclusion, Object Storage is not a replacement for Hadoop. It complements it by presenting low-cost storage alternatives to make processing more powerful. It gives as well flexibility in terms of data locality. Besides, it gives simple web services interfaces for access through RESTful APIs. On the other hand, GPFS remains the desired solution when it comes to fast fetching data for real-time processing applications.

B) Offline big data processing

Two of the most well-known and most utilized big data batch processing frameworks are Hadoop MapReduce [91] and Apache Spark [92].

In the early deployments, Groupe PSA has chosen the former, which as its name implies, consists of two distinct but closely related sets of tasks; the *mappers* and *reducers* (Figure 5.8). The mappers split data into chunks and process them in a parallel fashion to generate key-value pairs (tuples). As for the reducers, their role is to combine the mappers' output to get a smaller set of tuples. In addition, MapReduce was designed to move processing near the data rather than move the data to the processing. It implies collocating both the storage power and the processing power. Despite its high efficiency in terms of one-pass computations, MapReduce had numerous disadvantages. To improve the data lake sublayer performance and overcome the limitations imposed by MapReduce, the second framework (i.e., Apache Spark) has been considered. The following list lists some of the features that make Spark very useful and efficient:

- While MapReduce is recognized to be inefficient and inadequate for multi-pass computations such as iterative machine learning, Spark is proven to be very efficient in this regard [92].
- Spark is claimed to be 100x faster than MapReduce on memory and 10x on disk [92].
- To provide a fault-tolerant distributed memory abstraction, Spark implements the concept of *resilient distributed datasets* (RDDs) [93].

- Above all this, Spark offers ease of use and development through programming languages such as Java, Python, Scala, and R.
- Last but not least, Spark loads partitioned data into memory; therefore, there is no need to collocate the processing and storage.

In conclusion, MapReduce's usage today is reduced to the mere operations of backups and exports of HBase tables.

5.3.6/ DATA LEVERAGING AND SERVING

This layer takes the data produced by the lower processing layer (both data lake and speed sublayers), indexes it, and serves the result to the final users. In the following, we will first talk about data indexing and then data serving.

Once all the data has been collected, a layer of data virtualization is needed to simplify the accessibility of the data's underlying complexity. In his book, "Data Virtualization for Business Intelligence Systems", Rick F. van der Lans defines 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."* [94]. The data virtualization software aggregates structured and unstructured data sources through a dashboard or visualization tool for the virtual viewing. The software allows for the discovery of metadata about the data but hides the complexities associated with accessing disparate data types from different sources. Also, note that data virtualization does not replicate data from source systems; it only stores metadata and logic for display integration.

To index data, two different NoSQL databases must be considered: real-time random read/write and batch queries [95]. In the currently deployed architecture, Groupe PSA has opted for Apache HBase [96] and Apache Phoenix [97]. HBase is a distributed, column-oriented NoSQL database modeled after Google BigTable [98]. As for Phoenix, it is an open-source SQL engine that provides low-latency queries for data stored in HBase.

We mention that there is no precise definition of NoSQL. This term was originally used to define a new non-SQL derivative of the conventional RDBM systems[7] [99]. Recently, NoSQL represents a new class of big data DBMS. So, it might also refer to "Not Only SQL" (to indicate that it can also support SQL-like query languages). NoSQL databases have three common characteristics: high scalability, data replication, and schema-lessness. First, to deal with the large increasing size of data, NoSQL databases are distributed over clusters of machines. Second, to ensure redundancy and increase the system reliability in front of losses, data is replicated on different machines. Third,

[7] RDBMS: Relational Database Management System.

unlike traditional RDBMS, in NoSQL databases, no schema needs to be followed. Finally, we also mention that there are four basic types of NoSQL databases: key-value, column-based, graph-based, and document-based [95].

To make the leveraged automotive data available to final users, Groupe PSA exposes the data in two modes: RESTful APIs [100, 101] and MQTT. 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 shown in Figure 5.4, note that 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 all sorts of activities, improving new products/cars, customer services, etc.).

5.4/ AUTOMOTIVE APPLICATION'S EXAMPLES

In addition to its importance and usefulness to vehicle manufacturers, automotive data can also be very useful in a large range of customer services. In the following, we briefly describe some possible connected vehicle applications, and also provide some actual services that are currently deployed by Groupe PSA.

- **Electric and self-driving vehicles:** nowadays, self-driving and electric vehicles are very hot topics in the auto industry. In this context, the collected automotive data (related to vehicles themselves, the weather, etc.) can be properly leveraged to meet the requirements of these interesting technologies and provide them with essential services. First, regarding electric vehicles (EVs), a wide range of services related to their battery life can be considered. We mention as examples, battery preconditioning, services that allow EVs to have extra battery charges, information about nearest battery charging stations, and route suggestions that take into account both current remaining charge and charging stations. Second, as for autonomous vehicles (SDVs), based on the collected automotive data, multiple driving assistance features can also be offered to them. For example, autopilot in heavy traffic or highways, lane-keeping, active city brake, and parking assistance.

- **Vehicle management:** connected vehicles (regardless of their type; electric or fuel-based engines) send a large variety of data, including diagnostic-related ones (e.g., error codes, mileage or kilometers remaining before the next service, etc.). This helps the maintenance team to get relevant information before the admission of vehicles.

- **Mobility management and safety:** by fusing and analyzing the collected automotive data, drivers can be offered a faster, safer, and fuel-economic mobility experience. For instance, using automotive data, heavy traffic zones can be identified and efficiently avoided, hence reducing pollution, time, and fuel consumption. Applications of this category can also detect and alert in real-time drivers about potential dangers such as black ice, fog, rain, and accidents (prevent potential collisions).
- **Fleet management:** this automotive service can enable fleet owners (such as delivery cars/trucks and leasing companies) to have, in real-time, all the critical information about their vehicles (real-time tracking). This way, fleet managers can better organize and follow their vehicles to ensure responsible use, improve efficiency, reduce costs, and ensure safety.

The remaining two subsections describe two examples of services that are currently deployed via the PSA platform: *eco-driving service* and *weather service*. In the next section, we make an *"implementation focus"* on the eco-driving application by providing implementation details.

5.4.1/ ECO-DRIVING

The goal of this service is advising drivers and helping them improve their driving (making it eco-friendly). The current assessment of customers driving style adopted by Groupe PSA is represented by scores. The global eco-driving score and its sub-scores are divided into two types: (a) the driving style, namely a score on the driver's behavior, and (b) the road profile; a score related to the selected itinerary. In brief, 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 pertinent environmental data (such as altitude, road's slope, and other customers driving). This way, the customer can be helped to improve its driving and manage its fuel consumption. It is worth mentioning that this service does not only reduce CO_2 emissions and fuel consumption [102], but it can also extend the life cycle of vehicle spare parts through the optimized usage of the entire vehicle. The next section will go through the specifics of this eco-driving application. More details on that, in the next Chapter 6 where the second contribution of thesis will be detailed.

5.4.2/ WEATHER SERVICE

To meet the requirements of this application which is in partnership with Météo-France[8], each connected vehicle of PSA captures the external ambient temperature, tags it with its

[8]The French National Meteorological Service: http://www.meteofrance.com

current location, and then sends the result to the PSA infrastructure. The latter correlates the received data and updates the map (Figure 5.9). 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.

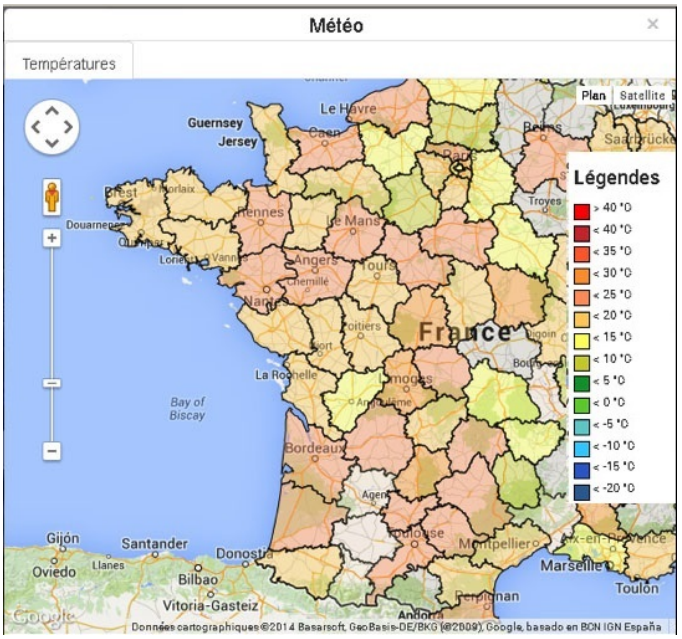


Figure 5.9: Weather service of Groupe PSA.

5.5/ Evaluation of the Architecture

In this section, we evaluate the architecture. We do that by considering two dimensions: The performance and the quality. First, evaluating performance translates into examining technical architecture behavior when facing fluctuations in the data flow, i.e., its scalability. We observe how PSA uses a reactive policy to deal with trends accompanying the rising number of connected vehicles. Second, the quality dimension consists of dealing with two sub-characteristics, which is the Quality of Data and the Quality of Service, as introduced in their hybrid quality evaluation approach [103]. The evaluation's goal is to support and justify the technological choices made in this architecture.

5.5.1/ Performance

Today, the automotive sector of Groupe PSA accounts for 1 million monthly active connected vehicles' users in France only. This number includes CVs whose owners have agreed to transmit their data to the infrastructure. Currently, they perpetually transmit data around **65 million messages** per day on average.

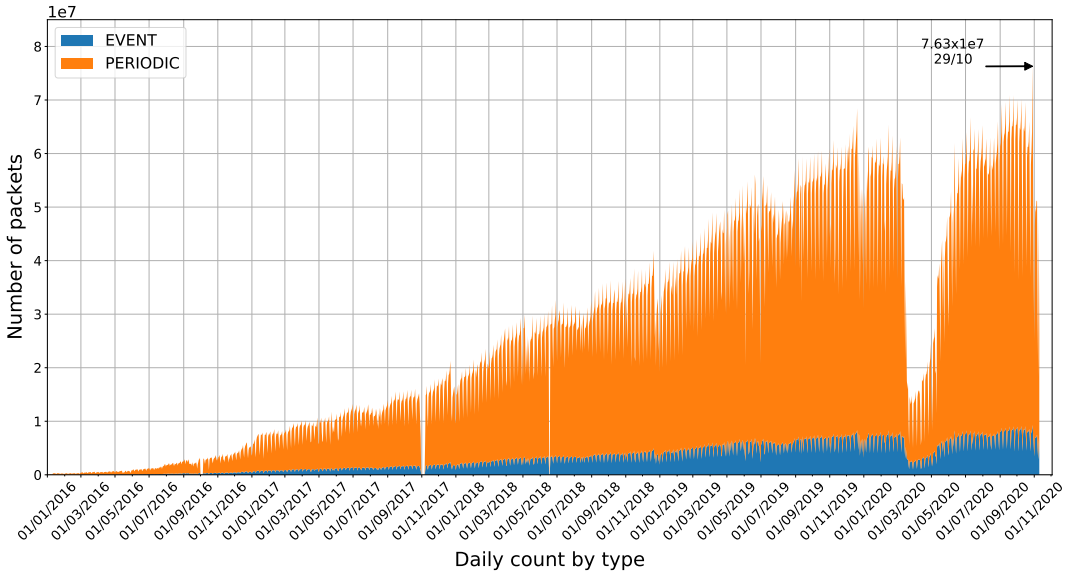


Figure 5.10: Evolution of transmitted CVs packets throughout the years.

Figure 5.10 shows the evolution of the number of PSA's connected vehicles throughout the years. The blue segment/layer on the bottom corresponds to eventual packets, and the orange one corresponds to periodic packets. Those two types of packets are complementary. As its name indicates, the periodic packets are recurrent sensor measurements embedded in the vehicles and later sent to the infrastructure. However, 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. In an Eco-Driving application context, a typical trip is composed of two eventual packets and several periodic packets depending on the trip length. An interesting observation is that the number of connected vehicles is on the rise. This phenomenon is explained by the trend of equipping our everyday objects with connectivity to propose to end-users new useful services and intelligence. Vehicles and transportation means are not an exception to this rule. The connected vehicle today proposes several services to drivers ranging from Connected Services, Infotainment, Safety & Security to Autonomous Driving. According to Verified Market Research[9], the CVs market will be worth $ 215.23 billion globally by 2027, with a compound annual growth rate (CGAR) at 14.56%. Groupe PSA was proactive to that trend by choosing and building throughout its CV platform, distributed, and scalable systems while investing in hardware.

Also, the abrupt discontinuities or gaps in the graph are due to occasional system upgrades or maintenance downtime. Those irregularities decrease in time due to the implementation of redundancy policies. Like many other organizations, PSA Group has to deliver business continuity. Disaster recovery plans have been implemented through

[9]PR Newswire: https://www.prnewswire.com/news-releases/connected-car-market-worth--215-23-billion-globally-by-2027-html - accessed on 11/16/2020.

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. In such a way, servers update one after the other without introducing any disruptions to the services. However, the sudden decrease in March and April of 2020 is not related to a technical problem. It is caused by the lockdown imposed in France as a sanitary measurement against the COVID-19's spread.
On the contrary, on October 29, 2020, we witness the complete opposite. A sharp spike in the number of packets sent to the infrastructure a day after the President of the French Republic announced a second lockdown. People rushed to stores to stock up and to perform their essential tasks.

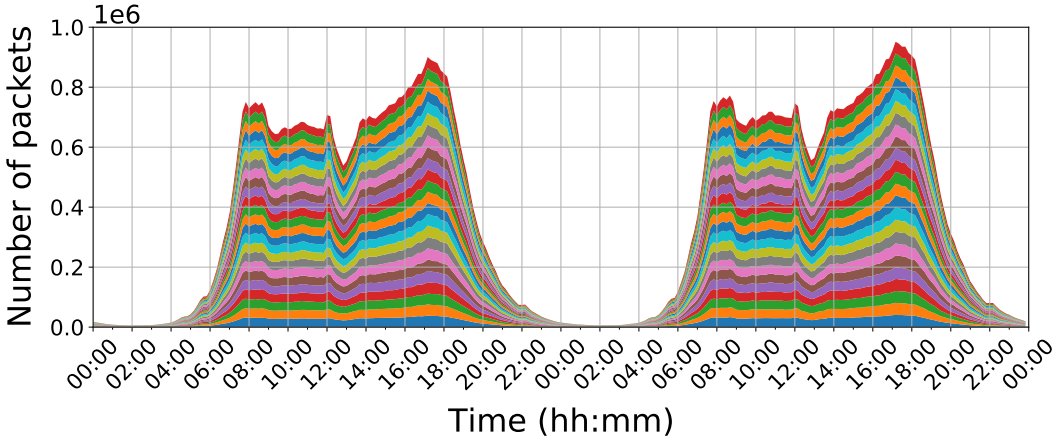


Figure 5.11: Number of packets transmitted per Kafka partition (over a period of 2 days).

Figure 5.11 shows the variation of the number of transmitted packets by the CVs in a period of two days. Every day, we observe approximately the same pattern with more or fewer data. Throughout a day, vehicles' generated data is not constant since vehicle usage varies and peaks in the morning and night. Those peaks correspond to people's movements when they leave for work and return from it, respectively. 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 used more or less their cars depending on if there was a lockdown period approaching. Likewise, on weekends and holidays, the transmission frequency is altered. Moreover, the colored segments/layers in the stacked area plot represent the content of 24-partition Kafka topics. Currently, our Kafka production environment is composed of 9 nodes. We configured topics of 24-partitions with a replication factor of 3. This partitioning totals to 72 partitions in total, which corresponds to a multiple of 9. Therefore, simplifying the messages' distribution for storage amongst the nodes. In addition to its highspeed transmission rates and data persistence, Kafka has a reliable

mechanism of distributing messages equally throughout the partitions; hence, throughout the servers/brokers.

Currently, during peak times (see Figure 5.11), in a 5-min interval, the infrastructure receives around 950K messages. That is translated to around 3100 messages per second. Therefore, to stress-test the architecture using Kafka, we batch messages of different sizes and monitor the number of sent messages in a second. Figure 5.12 shows that we can handle up to 7 times the load we have today with a batch size of 500k messages with the current state of the infrastructure.

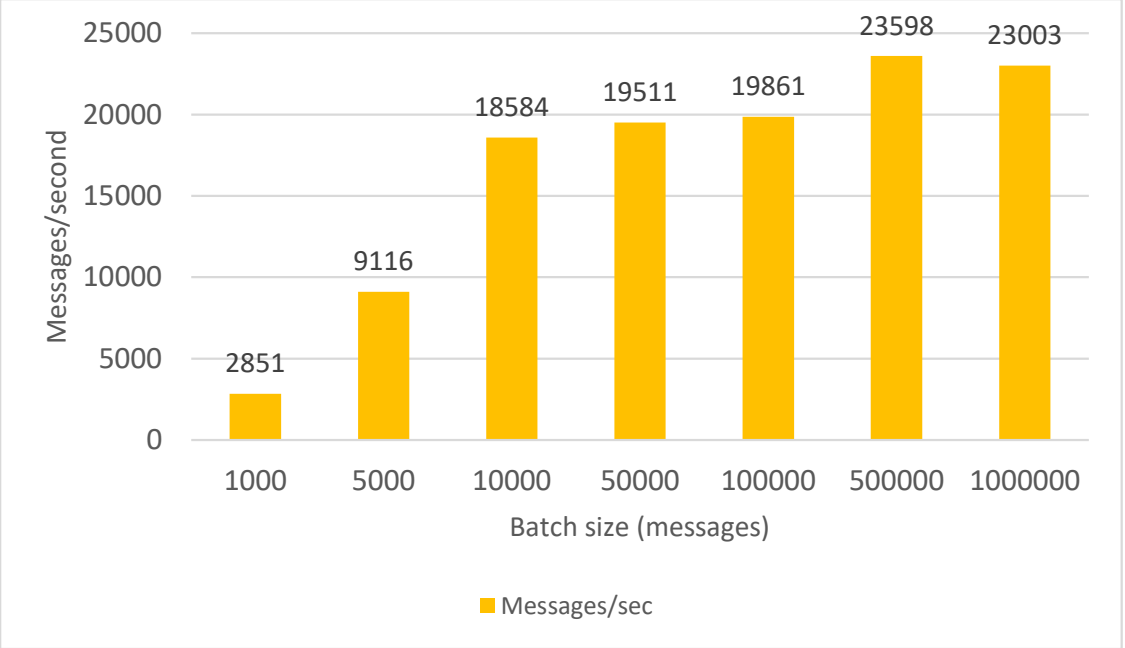


Figure 5.12: The number of records increments with the increase of the batch size.

In all previous scenarios, the architecture handled the fluctuations intelligibly. It remained available to deliver with no downtime registered, mainly due to the distributed frameworks used on all architecture levels, whether speed processing layer (e.g., IBM Streams, Apache Flink), storage layer (e.g., Apache Kafka) or batch processing layer (e.g., Apache Spark).

5.5.2/ QUALITY

A./ DATA QUALITY DIMENSIONS AND METRICS

Research shows that data quality is made up of many dimensions. Data quality dimensions are grouped into two groups by Lee et al. [104]: *intrinsic*, referring to attributes that are objective and native to the data, and *contextual*, referring to attributes that depend on

the context in which the data is observed or used. Relevance, added-value, quantity, credibility, accessibility, and reputation of the 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 to considering the intrinsic data quality measures [105]. Intrinsic data quality is consistently defined in the literature in four dimensions: accuracy, timeliness, consistency, and completeness. Before we proceed, we agree on the following definitions found in the literature:

- **Accuracy:** inspects the level of how well the reported information is accurate and resembles the real world's values and is therefore reliable.
- **Timeliness:** inspects the level to which extent the data can be considered as up-to-date. The literature suggests that timeliness is divided into two sub-classes: (1) currency, i.e., the duration since the record's last update, and (2) volatility, which inform about the update's frequency.
- **Completeness:** inspects the level to which records would be considered complete in content with no missing data.
- **Consistency:** inspects the level to which related data records remain consistent with the predefined structure and format.

Table 5.2 resumes the quality metrics for each of the quality dimensions above. Each metric can be calculated by applying its corresponding formula [106].

B./ Quality of Data: Evaluating the gathering layer

The data quality is best evaluated around two phases of the Big Data value chain, at the reception of the data and downstream after processing. Figure 5.13 shows the different stages of a framework proposed by Serhani et al. [103]. They present a 4-tier hybrid model for the Big Data value chain's quality assessment. Data quality is evaluated in both pre and post-processing. However, process quality evaluation happens between pre and post-processing and, at the end of the chain, where analytics occur, such as machine learning algorithms.

In this work's scope, we focus on the data quality at the preprocessing level, that is, the gathering layer after transmission from vehicles to infrastructure. The quality evaluation at the *'Processing & Analytics'* was not considered since it evaluates data from a functional perspective and not from an infrastructure perspective as what we aim

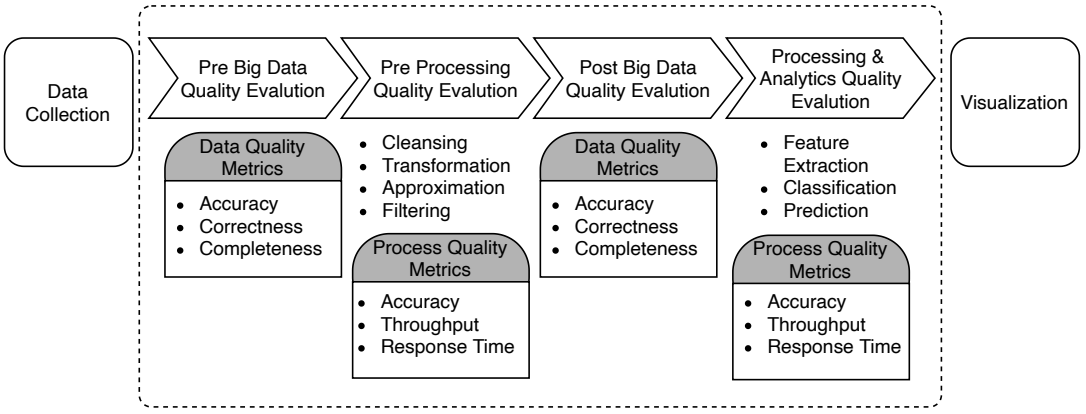


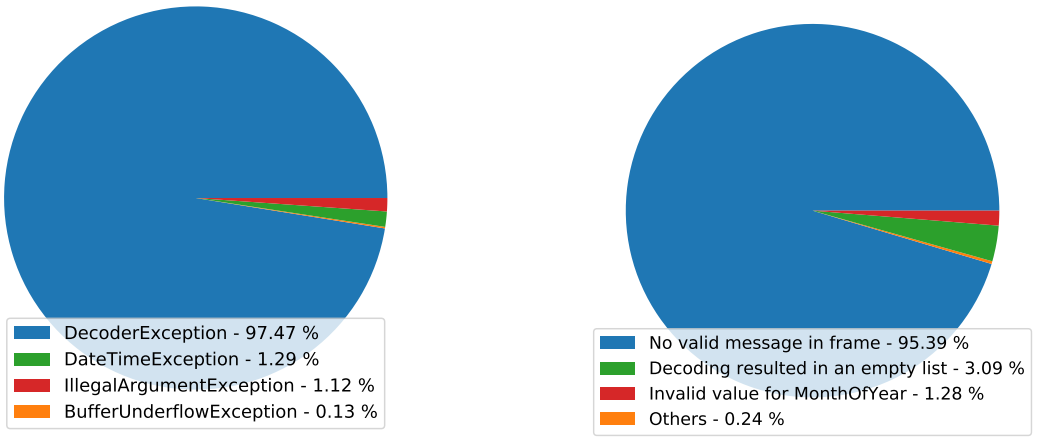
Figure 5.13: A hybrid model for evaluating Quality of Big Data value chain. Taken from [103].

to do in this work.
The preprocessing in this architecture can be divided into two steps, (1) the decoding of raw binary data arriving from Telematic boxes and (2) the preprocessing happening at a later stage at the beginning of the stream processing chain filtering inconsistent data. Therefore, we evaluate the two following inconsistency metrics for each of the previous steps: (1) serialization inconsistency and (2) content inconsistency.

In a typical 24-hour period, the PSA infrastructure receives 65 million packets, of which 130k raises an exception when processed. Therefore, the error rate is approximately 0.2%. This error rate corresponds to the first inconsistency metric related to serialization. To calculate this metric, we used the following formula from Table 5.2: $CNSM_a = \frac{numOfInconsistentValues}{totalValues}$.

For the content inconsistency metric, we used the Accuracy metrics $AM_a = \frac{numOfCorrectValues}{totalValues}$. The result was enormously satisfying with a low number of errors related to values' correctness since the first serialization filtered almost all the aberrant packets. We get a result of 99.9%.

In Figure 5.14, we see the different types of exceptions and the reason behind each, respectively. The graph in illustrates the distribution of the exceptions raised by the platform deserialization of the binary data coming from the vehicles. Four types of exceptions are found: *DecoderException*, *IllegalArgumentException*, *DateTimeException*, and *BufferUnderflowException*. From the doughnut chart, it is clear that the majority of errors originate from the *DecoderException* with 95.4%. The other three exceptions account for less than 3%. The reason behind that is that *DecoderException* class is the most generic exception class; therefore, errors are more prone to fall under this category.



(a) Error exceptions.

(b) Human-readable message errors.

Figure 5.14: Two pie charts showing the distribution of the exceptions and the reason behind each, respectively.

C./ Quality of Service: Evaluating the messaging system

To evaluate the Quality of Service (QoS), we choose to assess the communication layer that connects all architecture components thoroughly. Besides, the communication layer can be a decisive element when it comes to architecture performance. Our platform spans onto two separate data centers distant enough to consider applications' deployment locations. For instance, move data closer to the processing, particularly for real-time applications.

As described above, our Big Data architecture is composed of several interconnected layers. In every typical Big Data architecture, several components work together to collect, clean, process, and persist data. Therefore, those processes demand a robust, fault-tolerant ubiquitous infrastructure that takes the role of a messaging system that can dispatch all the needed data throughout the architecture reliably and efficiently.

In the following, we evaluate our choice of the messaging system at PSA Group, Apache Kafka. We test the performance of this system on different levels. Kafka connects several clusters of machines, each specialized with a particular task, as shown in Figure 5.15a, such as stream processing, batch processing, storage, and visualization platforms. The fundamental element of Kafka is the client instance running on those clusters. Clients can be *producers* or *consumers*. In this evaluation, we test data transmission performance from the producer and consumer clients running on different systems to the Kafka cluster, i.e., the Kafka brokers. We consider a Kafka cluster of four nodes deployed on one of the PSA Group data-centers (DC-A). The several services shown in Figure 5.15a are not but a subset of services. This test will evaluate Kafka's QoS by analyzing the *throughput* and *latency* from the Kafka clients to Kafka brokers. The services for which the communica-

tion client-server will be tested are the following: (1) HTTP servers taking on the job of decoding the transmitted data, (2) streaming servers incubating stream processing applications, and (3) visualization services of data to help decision-makers on getting better insights of 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 distant of more than 700 km: DC-A and DC-B. To test Kafka's client performance, we consider different parameters: the **throughput**, the **size of the message**, the **batch's size**, and the **number of messages** sent or received.

We have not imposed any throughput constraints for our load and stress tests so that the Kafka client can take full advantage of the infrastructure's capabilities. Moreover, we set the size of 1 KB for the sent messages since it would be the closest to a CV application's real-world scenario. The test consisted of sending one million 1 KB messages with no limitations on the throughput.

The graph in Figure 5.15b illustrates the throughput of several applications to the Kafka brokers deployed on DC-A. The columns show that applications, when deployed on DC-A, produce higher throughput, and this is due to their co-locality in the same datacenter. Likewise, the latency is lower when services are closest to the Kafka brokers, see Figure 5.15c.

Different compression types were used and compared side by side, as illustrated in Figure 5.16. Again, the Kafka client was run in DC-A, which explains the higher throughput compared to DC-B. The histogram shows that the snappy compression algorithm wins in both data-centers' data transmission.

In conclusion, applications or platforms that need high throughput and low latency data transmission, such as stream processing applications, were deployed in the data center closest to where the data is stored. However, for applications such as batch processing or visualization tools that build graphs about daily, monthly, or yearly data, can be placed as separate data. Although we choose data and task placement preference, the multi-datacenter deployment is always considered for the disaster recovery plan.

5.6/ DISCUSSION

Despite its efficiency and robustness, the presented architecture is far from being optimal and still suffers from several shortcomings. For instance, recall that in the original MQTT protocol, the role of the broker is limited to data forwarding from connected vehicles (publishers) to the automotive infrastructure (subscriber). This causes the infrastructure to be continuously overloaded with huge amounts of useless automotive data. To overcome

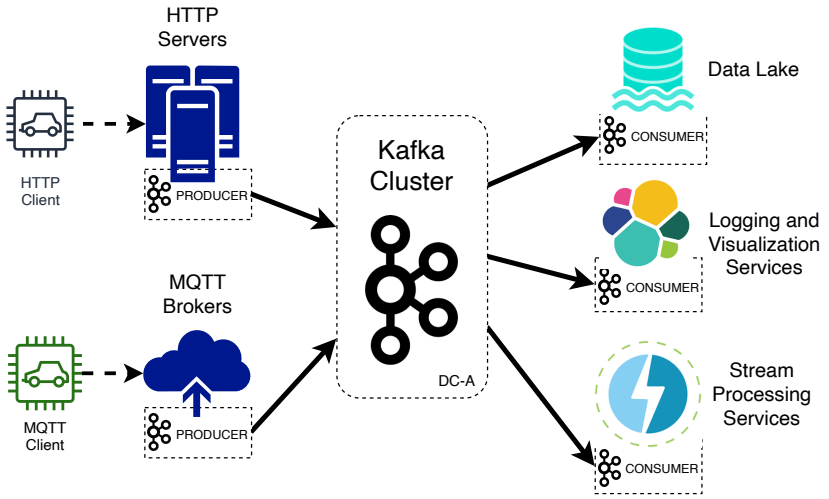
this issue, a new variant of MQTT dedicated to connected vehicles (called MQTT-CV) can be considered [107]. The advantage of this new variant is that it gives the broker the ability to process, filter, and feed the infrastructure with only data that satisfies its conditions and requirements. Note also that the presented architecture is unidirectional (vehicle-to-infrastructure (V2I) communications). However, as we have seen, for some automotive applications and services (such as self-driving/electric vehicles, safety, and vehicle management), the infrastructure must be able to communicate with its connected vehicles (infrastructure-to-vehicle (I2V) paradigm). Finally, we point out that there is a new work regarding the redesign of the speed processing sublayer, where both the adopted structure and technologies are reviewed. The goal is to switch to a more scalable and modular platform of microservices.

5.7/ CONCLUSION

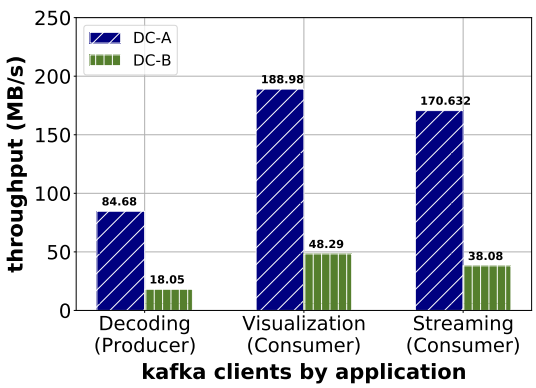
This chapter aimed to propose a complete (efficient, scalable, secure, and robust) model that is concerned with managing and protecting big automotive data throughout its life cycle (i.e., from data sensing, gathering, storing, processing, to the final data leveraging step). Although numerous researches have been done and still undergoing on efficient big data storage, processing, and leveraging, to the best of our knowledge, not much research has been done regarding this axis. To bridge this gap, in this chapter, we have thoroughly described how Groupe PSA (the second largest car manufacturer in Europe) gathers, stores, processes, and leverages its big data. These different tasks, along with the technologies and products that ensure them, were presented in a structured manner using the actual layered big data managing architecture deployed by PSA. In addition to the architecture description, we have also presented some potential automotive applications and gave some actual services that are presently provided by the PSA platform.

Table 5.2: Data quality metrics and dimensions. Taken from [103].

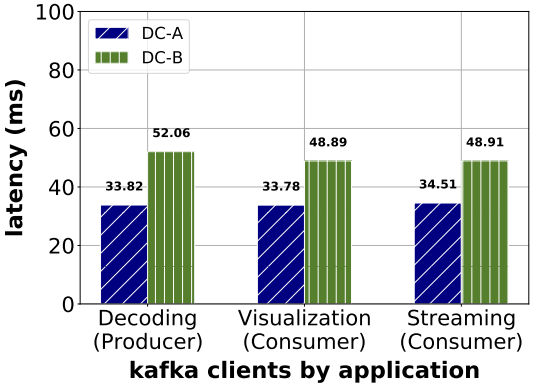
Formula		Description
Timeliness metrics		
TM_a	$= 1 - (CM_a/VM_b)$	1 - Currency/Volatility
TM_b	$= numOfProcessedRecs/totalRecs/timePeriod$	Percentage of the completed processed records within a time limit
Currency metrics		
CM_a	$= currentTime - updateTime$	Time of update
CM_b	$= updateTime - storageTime$	Difference between time of update and time of storage
Volatility metrics		
VM_a	$= ConstantTimePeriodValue$	Time length for which data remains valid
VM_b	$= (storageTime - updateTime)/totalTime$	Volatility: (time of data – time of update)/total time
Accuracy metrics		
AM_a	$= numOfCorrectValues/totalValues$	The ratio between the number of correct values stored and the total number of values.
AM_b	$= AvgUsrResponse$	User survey
Completeness metrics		
$CMPM_a$	$= numOfEmptyValues/totalValues$	The ratio of the number of empty of null values over the total number of values.
$CMPM_b$	$= AvgUsrResponse$	User survey
$CMPM_c$	$= actualTotalSize/expectedTotalSize$	The total size of the stored records over the expected size of the data
Consistency metrics		
$CNSM_a$	$= numOfInconsistentValues/totalValues$	The ratio of the total number of inconsistent values over the total number of values
$CNSM_b$	$= numOfViolations$	The total number of values violating constraints and rules



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



(b) Kafka client throughput vs. servers/applications.



(c) Kafka client latency vs. servers/applications.

Figure 5.15: Throughput and latency of Kafka clients when deployed on DC-A and DC-B.

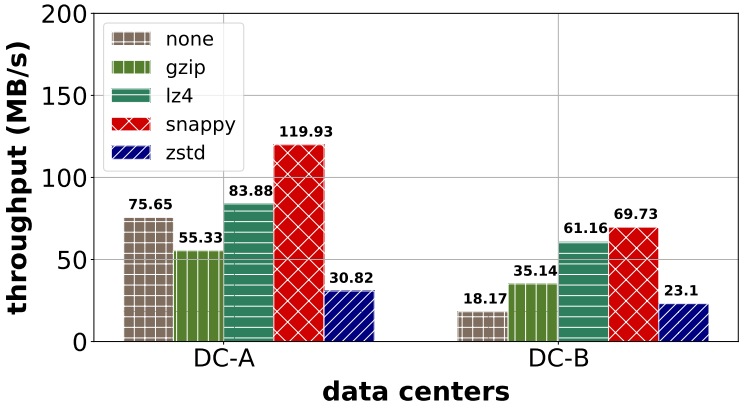


Figure 5.16: Load testing of the messaging system by transmitting messages to different data centers using different compression algorithms.

6

Adaptive Resource Allocation in Big Data Automotive Infrastructure

"The measure of intelligence is the ***ability to change****"*

Albert Einstein

Automotive applications, based on connected vehicles (CVs), are emerging as promising applications in several domains ranging from road security applications to smart city applications. CVs can collect, thanks to onboard sensors, up to 170 different readings (outside temperature, etc.), which are sent to a central big data automotive platform. To cope with this huge amount of data, researchers and engineers developed new paradigms, such as stream computing, in which a set of operators continuously processes data (i.e., the notion of stream). Hence, a streaming application can be modeled as a directed graph where vertices are operators' instances, and edges are data streams (i.e., continuous series of tuples). The central challenge in deploying streaming applications is the way to map operators' graph, representing the application, to available physical resources in order to enhance the application's performance (increasing the throughput, reducing the processing time). In this chapter, we target this issue by showing that the approach based on inherent data parallelism does not necessarily lead to the best performance for all applications. In fact, through a real-world application (i.e., the Eco-Driving service), we show that a mapping based on insightful analysis of the target application's specifics and the infrastructure's features can significantly improve the application performance, leading thus, to a noticeable impact. Our real experiments show that our proposal improves over the straightforward approach by about 4% in terms of throughput. This improvement allows PSA's infrastructure to handle nearly 800 thousand of additional vehicles over the expected 20 million CVs by 2025.

This work has been published in IEEE Symposium on Computers and Communications (ISCC), 2019.

6.1/ INTRODUCTION

Nowadays, connected vehicles (CVs) with sensing and communication capabilities are a reality. In fact, since 2015, all manufactured cars of PSA (Peugeot-Citroen French cars manufacturer) are connected. They can collect up to 170 readings ranging from external temperature to the angle of the steering wheel and sending them to a central infrastructure through 4G/5G communication channels. CVs raise many extraordinary business opportunities in several domains, including services for safe driving, security, etc. The success of all those services depends on the infrastructure's ability to gather, process, and derive adequate information in real or near real-time in order to deliver it to the requested application. For instance, Groupe PSA is developing a set of services provided to its clients, called *"Eco-Driving"*. This service needs first to collect information about the driving behavior of the client, the infrastructure gathers then this information and derives a trip score with constructive advice if needed; more details about the implementation of this service are provided in Section 6.3.1.

CVs generate, in a continuous way, such a massive amount of data that traditional data processing approaches are not able to efficiently support. For example, PSA infrastructure expects to support simultaneously up to twenty million vehicles. Hence the infrastructure is facing many challenging issues: significant data volume, generated continuously, in addition to the fact that usually, its feedback (i.e., processing results) must be in real or near-real-time.

To overcome these issues, new approaches, based on big data technologies, have emerged and adopted. They make use of the *stream computing paradigm*. In this paradigm, data is processed continuously through elementary operators. A stream application is organized as a directed graph: vertices are operators instances, and edges are data streams, as shown in Figure 6.1, where each stream is an infinite sequence of data items, and each operator ingests data objects from incoming streams and outputs data objects on outgoing streams.

The design and the implementation of a stream application consist of two steps:

1. **Application graph modeling**: In this first step, the application is modeled as a direct graph in which the sources are the application's inputs streams, and the output is the expected results. The vertices are data operators (filtering, aggregation, etc.) that operate on intermediate data streams (i.e., graph edges). The number

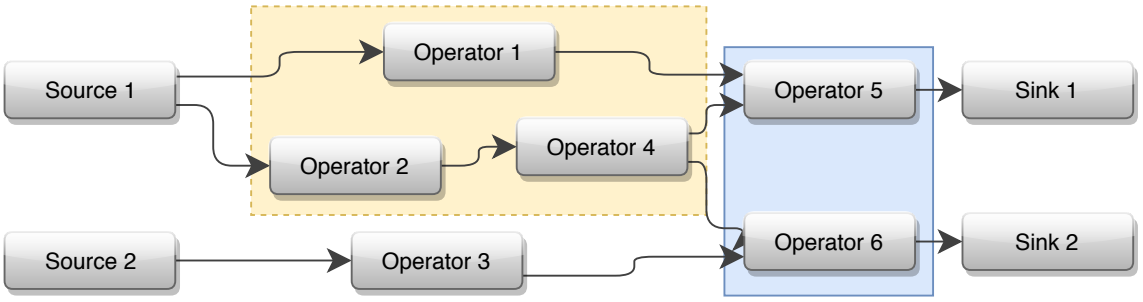


Figure 6.1: A streaming application with fused colored zones

of operators, their nature (single-input/single-output, multiple-inputs/single-output, etc.) as well as their complexity (computational, intermediate storage, etc.) is dependent on the target application requirements.

2. **Graph mapping**: Once the application streaming graph has been defined and validated (i.e., which conforms to the expected results), the second step consists in its *"mapping"* on the available physical, usually distributed, infrastructure. This mapping takes into account the target application requirements in terms of performance (increasing throughput, reducing processing delays, etc.), as well as the available resources of the target infrastructure. This process is known as *"resource allocation process" (RAP)*.

Whereas the first step requires a particular modeling effort from the application developers, the second step is almost left to the streaming engine (i.e., IBM Streams[108]). The latter inherently produces parallel executions of the operators whenever it is possible; i.e., operators are deployed on separate hosts and run simultaneously. For instance, in Figure 6.1, colored boxes represent the corresponding hosts.

Considering a practical application of CVs (Eco-Driving service deployed in Groupe PSA), we focus in this chapter on the second step. Our objective is to look for any possible improvement over the straightforward automatic approach because, in the automotive application, any apparent *"tiny"* improvement in the infrastructure performance will lead to a high impact on the overall application, in particular on the number of concurrently supported vehicles. The question was: *"does the automatic approach (i.e., inherent operator's parallelism) always derive the better RAP with regard to the infrastructure performance?"*

To answer this question, we conducted an insight analysis of both the applications specifics and the available infrastructure features. Based on this analysis and some infrastructure parameters measurements, specifically host's communications delays, we have proposed a new RAP which performs operator fusion (see below for details) based not only on intrinsic parallelism but also includes intercommunication delays. The ob-

tained experimental results show an improvement of our approach about 4% over the straightforward approach in term of infrastructure throughput saving. In term of application benefits this improvement allow PSA infrastructure to handle roughly 800 thousands additional vehicles over the expected 20 millions vehicles.

The rest of the chapter is organized as follows: Section 6.2 reviews the basic notions of streaming applications. Section 6.3 exhibits the real-world service of Eco-Driving and its implementation as a streaming application. Next, two strategies of operator placement are reviewed, tested, and assessed: the no fusion and auto-optimization strategies. Later, an enhancement that takes into consideration the specifics of our application is proposed, followed by experimental results to validate our work in Section 6.4. Finally, we end with a conclusion and a brief discussion about future works in Section 6.5.

6.2/ Speed Processing Sublayer

The previous Chapter 5 detailed the actual architecture deployed at Groupe PSA for big data analytics applications leveraging the data acquired from connected vehicles. We refer back to it in this section to highlight the technology behind the processing layer where the real-time applications run.

The big data managing model of PSA consists of six layers: (1) *data sensing*, (2) *data gathering*, (3) *data queuing*, (4) *device and referential data management*, (5) *data processing (with two different processing modes or sublayers)*, and finally (6) *data leveraging and serving*. The layered architecture is shown in Figure 5.2. All layers except for the data sensing layer (layer 1) are under the governance and ownership of Groupe PSA, whereas the data sensing layer is related to both the Groupe and its partner brands.

We give a brief description of the data processing layer where the real-time processing is being executed—the core layer of this work.

In the data processing layer, two sublayers exist, the speed sublayer and the batch sublayer, as shown in Figure 5.2. While the former is responsible for processing the continuous data streams (online processing), the latter stores and processes historical automotive data with offline processing jobs (such as machine and deep learning techniques). Following, we review the speed sublayer where the real-time applications are deployed. Moreover, we discuss the framework used at this level and the challenges we faced when deploying an application.

In this work, we are interested in the speed processing sublayer due to our application requirements. Nevertheless, before speaking about the needs, we detail the main component in the speed sublayer. This sublayer is responsible for two main tasks: *data stream acquisition* and *data stream processing*. While the data stream acquisition deals

Table 6.1: A brief description of InfoSphere Streams main components

Component	Description	Our work
Tuple	Typically, the data in a tuple represents the state of something at a specific point in time. An individual piece of data in a stream.	Our tuples are composed of 26 variables either sent by the cars to the infrastructure or data enriched from our databases about the car itself.
Operator	An SPL operator manipulates the tuple data from the incoming stream and produces the results in the form of an output stream.	More than forty operators are used in our case, mainly distributed between source/sink operators and processing/transformation operators. Refer to Figure 6.3 to see a simplified diagram of the application.
Processing Element (PE)	Execution units created after the compilation of a stream processing application, which represents concretely the operator and streams relationships that make up its dataflow graph.	The challenge of getting the right number of PEs are tackled in the next section.

with retrieving fresh automotive data from the lower queuing layer and delivering it to the stream processors, the data stream processing incubates stream processors that will retrieve the necessary referential data then run accordingly the stream processing applications.

Stream computing is a new programming paradigm that is designed for distributed and parallel processing of unbounded data streams. This paradigm is often confused with real-time processing, which requires a response within a certain time span. We can characterize a stream processing application by the following points:

- Data items are processed as they arrive (i.e., online).
- Events are time-based. Typically, every record is timestamped on creation.
- Operations are done in a data flow fashion/design.
- Every operation is done on one data element (or a small window of the recent data).
- An operation calculates something relatively simple.
- Each computation needs to be complete in (near) real-time to avoid congestions.

As depicted in Figure 6.2, streaming applications are executed as jobs on a *Stream Runtime Environment* (SRE), known also as *Instance*. Note that single or multiple processing nodes may be used to deploy a job.

Currently, at Groupe PSA, IBM Streams takes the role of the running Stream Processing Engine. As its name suggests, this platform is owned and supported by IBM. It

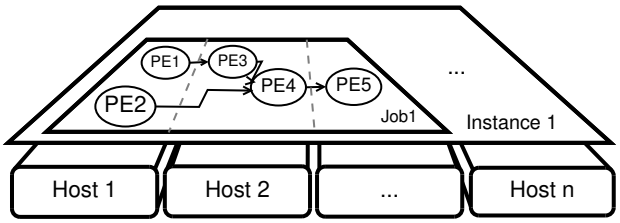


Figure 6.2: Illustration of the stream runtime environment.

enables the development and execution of applications that process information in data streams. In other words, it allows continuous and fast analysis of massive volumes of moving data. IBM Streams consists of a programming language and an integrated development environment (IDE) for applications and a runtime system that can execute the applications on a single or distributed set of hosts. Besides, it offers the IBM Streams Processing Language (SPL) interface for end-users to operate on data streams. SPL offers numerous operators, the ability to import data from external sources and export results outside the system, and an option for extending the underlying system with user-defined operators. Deploying stream processing applications creates a dataflow graph supported by the underlying runtime system. The main components of streams processing applications are tuples, data streams, operators, processing elements (PEs), and jobs. A more detailed look into those concepts shown in Table 6.1.

Moreover, IBM Streams provides several fusion schemes that specify how operators are fused into processing elements before compiling and running the application. The fusion scheme used can influence the runtime performance of the application. Therefore, in the following sections, we study closely two logically distinct schemes and assess them, subsequently examining its impact on the application performance and its limits. Next, we expose our proposition for enhancing the mechanism of process-operator allocation.

6.3/ A Case-Study Application

Many connected vehicle services or applications can be developed today as a streaming application and deployed in a centralized environment. The centrality allows the streaming application to process data from a myriad of sources and whereas the flexibility allows to scale up or down, adapting to the flow rate of data transmitted to the infrastructure.

Before diving into details, we recall that our aim in studying this use-case is to illustrate, throughout a real industrial application, among numerous others, our proposed approach. It has to be noted that our proposal can easily be used or extended to any other automotive application. More precisely, we discuss two placement strategies, i.e., proposed by the system and considered as state-of-the-art solutions. Then we compare

them to our proposal.

Our claim is that because of the huge amount of data in an automotive application, approaches that rely on the application's specifics as well as on the platform's features can improve over straightforward ones.

6.3.1/ Eco-Driving Service Description

In one of its numerous digital services, Groupe PSA provides a service known as Eco-Driving. Eco-Driving makes it possible to score a driver according to their behavior and the characteristics of their journey, in addition, it advises the customer based on their weak points so that they can improve their driving. Further, the global score of Eco-Driving and its sub-scores are divided into two parts:

- The driving style, namely a score on the driver's behavior
- The road profile, namely a score related to the itinerary selected

Table 6.2: List of computed criteria by category

Subscores Eco-Driving	
Drive Attitude	**Road Profile**
Acceleration	Speed/consumption
Braking	Slope
Engine speed	Cold Trips
Stop & Start	

Each category of the above is composed of several fields of information used to compute its corresponding score. The driving style, for instance, takes into consideration the acceleration, braking, engine speed and the number of occurrence of Start & Stop functionality. Further, the road profile category focuses on the average speed and fuel consumption, slope and cold trips. Refer to Table 6.2 for a summary of the attributes used to calculate each sub-score. Every score can have a value ranging from 0 to 10. These scores are accompanied by two labels, powered by internal configuration tables and set by the business team:

1. an appreciation message
2. a tip on how to enhance client's behavior

6.3.2/ Application Architecture

At the time of writing, our servers deal with about 400k trips daily composed of thousands of tuples. Those trips need to be processed, as fast as possible, and be delivered back to

the client so they can get to review their performance before failing to remember their driving experience. Therefore, a fast, reliable and ubiquitous technology is needed to meet those requirements. Hence, the Eco-Driving application was developed and implemented in the Speed layer of our architecture described in section 6.2 in the form of a streaming application.

Eco-Driving application is composed of PEs deployed on hosts that communicates with each other either through the network or directly through shared local memory, depending on the deployment model or strategy used. Our work is built upon the components of InfoSphere Streams[1]. Figure 6.3, visualizes a simplified version of the data-flow graph of the developed application, and it exhibits the main operators and their ties between each other. Figure 6.4, visualizes a screen-shot of the complete graph of the streaming application in the Stream Studio IDE provided by IBM.

The application consists of three regions, notably, the ingestion region, processing and post-processing region and finally the dissemination region. It is in the core part of the application where the heavy work is done. The calculations, being independent and consuming the same data as input, were distributed in a parallel mode to speed up processing. As a result, we had several operators in parallel and functionally separated. At the end of the processing, a barrier blocks the flow until all the results are ready to be grouped and forwarded downstream.

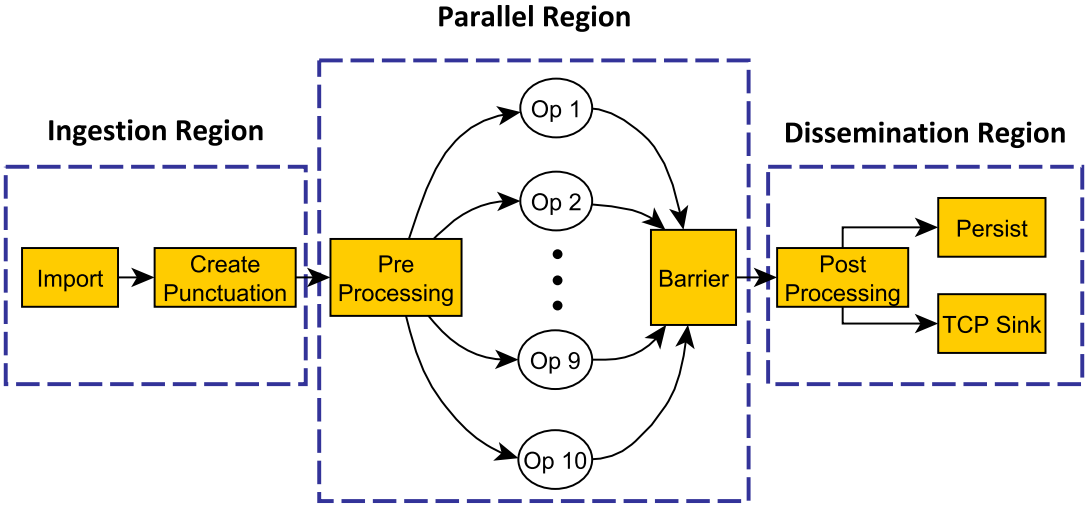


Figure 6.3: A simplified graph of the developed application

[1]The Eco-Driving application was developed using an earlier version of IBM Streams, InfoSphere Streams 3.2.1.

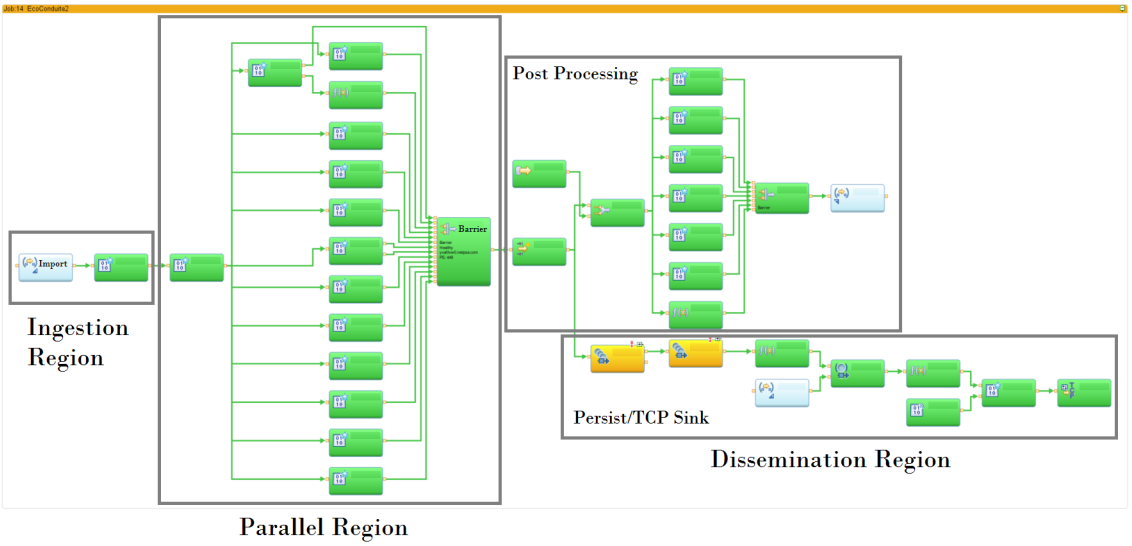


Figure 6.4: A screen-shot of the application in Streams Studio IDE

6.3.3/ DATASET

The dataset used in this project is a private time series dataset where data points are recorded every second by the vehicles' sensors and then sent through the network to Groupe PSA's cloud. Every 60 data points are aggregated into one packet, thus, reducing the flow rate to a packet per minute, where each contains information about the elapsed 60 seconds. The testing vehicle fleet consists of hundreds of Groupe PSA vehicles traveling all over the French metropolitan territories. Trips and data points schema are defined below:

Trips A trip consists of several packets/tuples depending on the length of the trip. For instance, a trip of 20 minutes will produce 20 packets/tuples along the journey, one every minute.

Schema Each packet or tuple is composed of 31 attributes used by the application to determine the EcoDriving score. The main attributes with their corresponding data types are listed in Table 6.3.

6.3.4/ FUSION STRATEGIES

InfoSphere Streams propose several fusion schemes, already introduced in section 6.2. Two of the available options that interest us in this work are:

Table 6.3: Tuples schema of the dataset used for experimenting with EcoDriving application

Attribute	Data type	Attribute	Data type
packetType	int16	...	...
virtual	uint8	consoTotal	float32
modePacketType	int64	kmTotal	float32
vin	string	conso	float32
vink	string	coordgps	string
tripId	uint64	srcsignalgps	int32
timestamp	string	freinurgent	int32
counter	int32	contactveh	int32
altitude	int32	nivcarb	int32
tempHuile	int32	etatmoteur	int32
speed	float32	cap	int32
rpm	float32	stopStartFn	int64

- **No fusion**: each operator will be assigned to a PE. Hereafter, we refer to this approach as the *Straightforward Approach*.

- **Optimized fusion**: the compiler will figure out how to best fuse operators to be hosted by one or more PEs. Hereafter, we refer to this approach as the *SPE built-in Approach*.

In the following sections, we develop each one of the enumerated approaches above. Then we test and assess them before proposing our enhancement.

Straightforward approach Since our application was parallel by design, the first call was to use the no fusion option. That is, assign a PE for every operator and distribute the operators in the parallel region to all given resources. As a result, data processing will become simultaneous and independent and ultimately introduce a gain in time and in performance.

SPE built-in approach The second placement scheme consists of applying profile-driven optimization. Before moving forward to results, we explain this optimization briefly. When operators are not fused into PEs, they communicate via the transport layer, denoting that every tuple must be serialized, transmitted over the network, and deserialized

on reception. However, when fused, they communicate through function calls, and tuples are passed from one operator to the next using memory references. Therefore, this could significantly reduce the cost of communication and improve both latency and throughput.

The fusion process can be automated by using the compiler's features to optimize the fusion based on a profile. In this mode, the compiler will select the combination of the operators to be hosted by one or more PEs, while maintaining the constraints specified by the user. The process is called fusion optimization. It requires a step for profiling wherein the application is first to run in profiling mode, to characterize the resource use of the CPU consumption and data traffic for each of the operators making up the application. In summary, there are two main steps involved in performing profile-driven fusion optimization: profiling (run-time) and optimizing (compile-time).

What characterizes a practical optimization approach is a robust profiling framework. IBM researchers [109] describe rigorously the profiling mechanism implemented in InfoSphere Streams in their published work. In short words, it consists of three main steps, namely code instrumentation, statistics collection, and statistics refinement. Code instrumentation is used to inject profiling instructions into the processing elements generated at compile time. During the runtime process, statistics collection executes those instructions to collect raw statistics on operators' communication and computation characteristics. Statistic refinement consists of post-processing the resulting raw statistics into well-formatted statistics suitable for consumption by the fusion optimizer.

Those steps were executed automatically after running the compilation. As an input, we provided two parameters for the statistics collection step: sampling fraction $s = 0.001$ to collect metrics once in every $1/s = 1000$ tuples and a reservoir size $S = 500$ that will keep a random subset of 500 values from the metric values collected, as recommended by the platform.

The resulting fusion scheme grouped all operators in one PE; as a consequence, the flow rate went up to about eight times compared to the no fusion scheme.

6.3.5/ PROPOSED APPROACH

The developed solution of Eco-Driving service should be able to sustain any future load possibly produced following the expected boom in the market of connected vehicles in the upcoming years. Therefore, the current version of the project, with its default allocation strategy, i.e., the no fusion strategy, was deemed to be unsatisfying. The likely rate of the data flow for a similar application is expected to be higher than what it is currently.

Based on what was discussed earlier, we propose an alternative approach that considers the strong points of both previous approaches and takes into consideration the

particular features of the considered application. Hence, two main particularities could be identified:

- the application architecture is composed of three functionally different and independent regions

- the composition of each trip is made of several tuples that are aggregated, processed, and ejected downstream after the reception of the end-of-trip signal.

As a result, we proposed a fusion strategy conform to the three-tier architecture shown above in Figure 6.3.

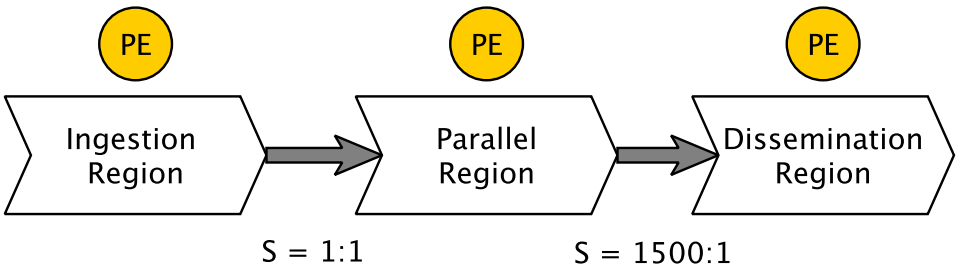


Figure 6.5: An illustration showing the selectivity between the regions

First, the ingestion and the parallel region are decoupled; therefore, the processing operators become independent of the ingestion process and its underlying threads. In such a way, we take advantage of what is known as the pipeline parallelism: the concurrent execution of an *operator* 1 and an *operator* 2 handling consecutive data objects, as shown in Figure 6.6. Therefore, the application, simultaneously, processes the data in the parallel region while preparing and pre-processing the next in the ingestion region.

Second, the parallel region is separated from the dissemination region, where the data flow rate is reduced by a factor of 1500 after being processed by the parallel region, as shown in Figure 6.5. This phenomenon is due to the second characteristic identified earlier. We considered here that the average duration of a trip is 25 minutes, hence, the 1500 seconds. The *selectivity* of an operator is its data rate measured in output data items per input data item. For instance, in our case, the parallel region produces one output data item for every 1500 input data items and has a selectivity of 6.66×10^{-4}. By decoupling the last two regions, we improve performance by reducing the number of useless function calls between the two when they are merged.

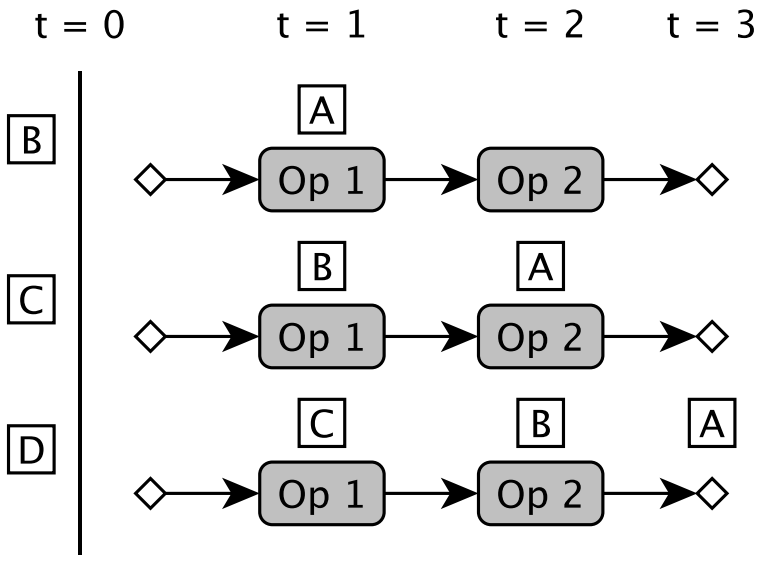


Figure 6.6: Data parallelism in stream applications

6.4/ EXPERIMENTAL EVALUATION

6.4.1/ STRAIGHTFORWARD APPROACH RESULTS

Assigning one PE to every single operator did not yield satisfying results. In an automobile company context and for this kind of application, to process five trips per second does not correspond to Groupe PSA's requirements for this project. Furthermore, the improvement it brought was negligible to what was already in place as a solution, i.e., a statistical sequential processing solution using the SPSS tool[2]. The application processed the same rate of trips with about three times the resource utilization rate.

6.4.2/ SPE BUILT-IN APPROACH RESULTS

Likewise, applying the profile-driven optimization to the application did not yield satisfying results. With the knowledge that operators deployed on different PEs communicate using TCP, two metrics were monitored to validate the hypothesis that the three main steps for transferring tuples: serialization at the source, transmission over the network and deserialization at the destination, are the logical explanation to the system's load and degraded performance.

[2]IBM SPSS Modeler is a data mining software application from IBM. It is used to conduct other analytic tasks. It has a visual interface which allows users to leverage statistical and data mining algorithms without programming.

To simplify the work done up to this point, we consider the following scenario. We suppose that the application incorporates in its graph three essential parts. In the beginning, we have the ingestion region, followed by the parallel region; finally, the dissemination region, as shown in Figure 6.7.

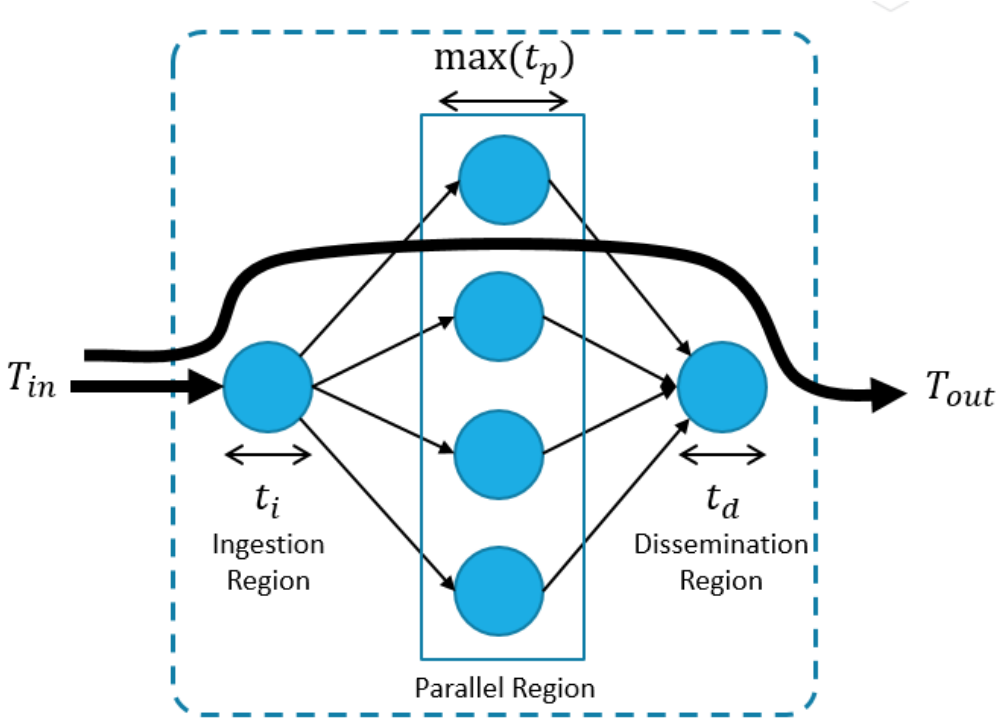


Figure 6.7: Illustration of the metrics used for the allocation process

Next, we define each one of the metrics used to study the application's behavior over different fusion strategies:

$t_{compound}$ corresponds to the aggregated processing time of each tuple belonging to a trip, and that passed through the whole graph, refer to (6.1). Therefore, in this variable we take into consideration the time of transmission of data items between operators as shown in Figure 6.7 by the thick black arrows.

$$t_{compound} = t_c = \sum_{k=1}^{N} T_{out_k} - T_{in_k} \tag{6.1}$$

where T_{out} and T_{in} correspond to the timestamp of the date when a tuple exited and entered the composite, respectively.

t_{unit} corresponds to the aggregated sum of three sub-metrics t_i, t_p and t_d, which simply represents the time the tuples progressed in every region, refer to (6.2). The max is used to obtain the longest execution time of an operator in the parallel region. Even

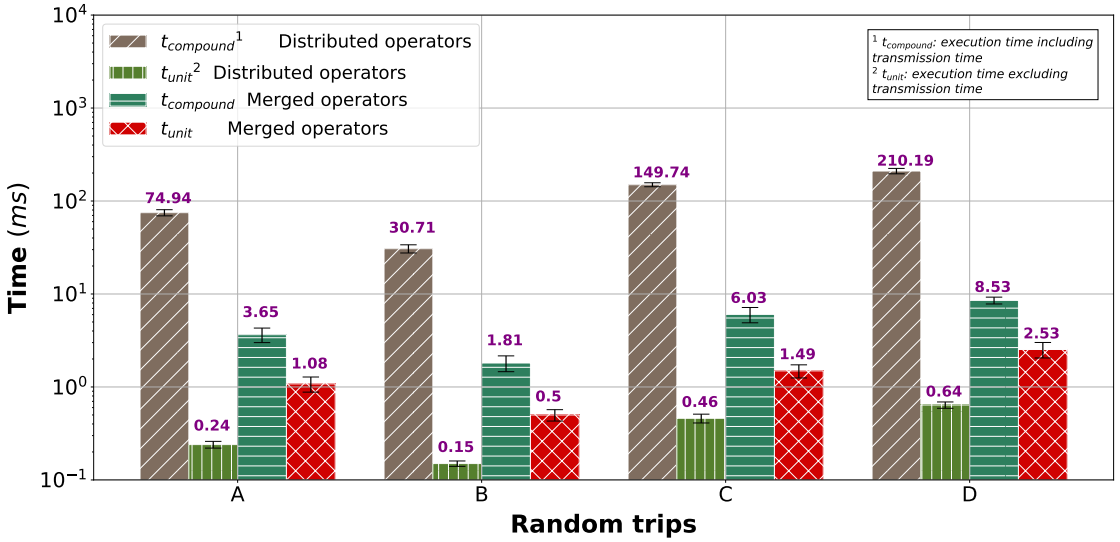


Figure 6.8: The application runs faster with merged operators rather than distributed ones, that is, one operator per PE

though we are considering the max of this region, the margin between processing times of operators is negligible. Contrarily, we include here only the computation time in operators as shown in Figure 6.7 by the double-headed arrows under the operators.

$$t_{unit} = t_u = \sum_{k=1}^{N} t_{i_k} + \max(t_{p_k}) + t_{d_k} \tag{6.2}$$

where t_i, t_p and t_d correspond to the time span needed for the tuple to finish execution in the ingestion, parallel and dissemination region, respectively. N corresponds to the total number of tuples for a given trip.

Normally, for the transmission not to be the reason behind the lagging flow rate, condition (6.3) must be satisfied:

$$t_{compound} = t_{unit} + \lambda_s \tag{6.3}$$

where λ_s is the time allocated by system calls.

Table 6.4 shows the results of the two strategies for 20 trips comparing them side by side. This number of trips was chosen randomly from the bigger set of 28 trips for layout constraints. The results show that there is a remarkable difference between the two processing times. Looking at the average of the results, we find that in the *No Fusion* approach there is nearly one second of difference between the real computation time, t_{unit} and the time aggregating processing time and the transmission cost along with it, $t_{compound}$. However, when fused the $t_{compound}$ decreases remarkably.

This huge difference is explained by the fact that the initialization of the packet to be

Table 6.4: The two metrics side by side calculated with two different deployment modes in ms

	No Fusion		Built-in optimization	
Trip	t_u **(ms)**	t_c **(ms)**	t_u **(ms)**	t_c **(ms)**
1	1.21	367.89	0.75	3.60
2	0.65	205.62	0.79	2.36
3	0.78	213.34	1.95	5.25
4	0.65	89.71	1.51	2.63
5	0.66	238.12	1.24	2.44
6	0.94	380.94	1.24	3.34
7	0.84	514.79	1.50	4.04
8	1.41	866.91	2.05	5.41
9	5.25	2572.06	7.73	55.61
10	2.09	1204.90	2.58	8.53
11	2.44	1267.26	2.54	8.95
12	2.01	936.99	3.67	7.73
13	2.33	1110.22	3.73	8.74
14	0.52	241.06	0.36	2.35
15	1.84	810.26	2.89	6.87
16	0.38	246.64	0.34	1.82
17	7.83	4306.97	99.92	113.90
18	0.32	191.12	0.49	1.52
19	4.28	1589.39	5.91	38.95
20	0.65	251.58	1.00	2.60
AVG	**1.85**	**880.29**	**7.11**	**14.33**

sent through the network consumes enormously compared to when it is in the same PE. Looking closely, most of the calculations in the parallel region had somehow low complexity algorithms. Leading us to the following conclusion: *when the time of processing of a tuple is significantly faster than the time of transmission, it is then not efficient to distribute the processing on different nodes*.

6.4.3/ THE PROPOSED APPROACH RESULTS

To apply the work described in 6.3.5, InfoSphere Streams provides directives to developers to co-locate or ex-locate operators in a PE. We used this feature to our convenience to divide our application into the three regions as shown in Figure 6.3. The application was ran 12 times with a sample of 28 trips for the four scenarios. The results and the corresponding standard deviation are shown in Figure 6.8.

The tests were divided into four categories: (a) *Distributed and PE-based processing time,* (b) *Distributed and operator-based processing time,* (c) *Merged and PE-based processing time* and (d) *Merged and operator-based processing time*. Where PE-based processing time is the computed value of $t_{compound}$ and operator-based is the computed value of t_{unit}, presented in section 6.4.

The four scenarios are the following:

1. *Distributed and PE-based processing time*. In this scenario, all operators are exe-

cuted by a single process and occupy one PE. Technically, PE-based processing time is calculated by starting a timer when the first tuple of a trip enters until the last one is processed, signaling the end of the trip. In that way, everything that happens in between the execution of two consecutive tuples is included. To do so, we considered one operator from the parallel region for extracting the processing time of the application since all operators in that region execute concurrently and almost equally and that the parallel region accounts for the most resource-consuming part. Moreover, the significant part of the execution is supposedly spent on the transmission of tuples between processes/operators, and the processing time is negligible. The $t_{compound}$ metric was used in this category.

2. *Distributed and operator-based processing time*. In this scenario, all operators here as well are distributed. Operator-based processing time is the actual code execution time running inside of an operator for every tuple in the trip. The timer starts at the reception of a tuple and pauses when the processing ends. It later resumes after receiving the next tuple. The result is the sum of the elementary processing times. Therefore, we calculated in every operator the execution time, and then we aggregated this time by calculating the t_{unit}. For the time elapsed in the parallel region, t_p, we selected the slowest operator as a reference as it is the extreme case.

3. *Merged and PE-based processing time*. When merged, all operators coexist in one PE/process. Therefore, the computation of the $t_{compound}$ not only included the processing time in every code section of an operator but the system and function calls occurring in the inter-operator communication in the process.

4. *Merged and operator-based processing time*. We calculated the processing time inside of every operator and aggregated those values by calculating the t_{unit}.

Moreover, the processing time of a trip is proportional to the size of a trip. Therefore, comparing trips side by side is relevant and the difference in the processing time is only due to the different number of tuples forming a trip as shown in Figure 6.9.

Finally, Figure 6.10 shows the throughput achieved from different fusion strategies, relative to our proposed enhancement. The results show that our work performs 10, and 1.04 (4%) times better compared to *No Fusion* and *Built-in optimization*, respectively.

Having all that knowledge, we were able to pinpoint a particularity of the scenario at hand, which contradicts the common beliefs or one's intuition. Being that operators were embedding simple algorithms/calculations, the processing time was fast enough to make the upstream operator, which is ingesting the data into the parallel region of operators,

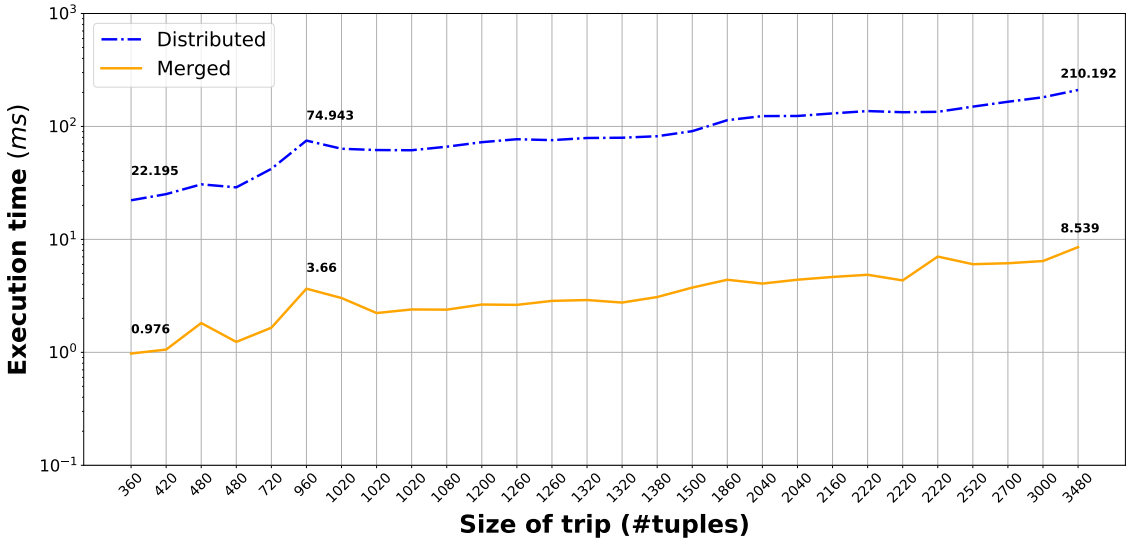


Figure 6.9: The merged fusion strategy performs better than the distributed one over randomly chosen 28 trips sorted by size

to become a bottleneck and slow down the traffic. Moreover, the operator was mainly serializing and transmitting, through the local network, every copy to its downstream operators whose numbers happens to be 10. Refer to Figure 6.3, for a general overview of the application graph of operators. Therefore, parallelizing and distributing operators onto different PEs was not the optimal answer to the problem due to the fact that when multiple trivial calculations comes into play, if their execution time is less than the transmission itself, then parallelism is inappropriate but sequential processing is.

6.5/ CONCLUSION

In this chapter, we have proposed an enhancement for operator placement strategies. We have used real-world application, Groupe PSA's Eco-Driving service, to showcase how built-in optimization is generic and could sometimes propose a less efficient optimization. Conversely, when specifics are taking into account, we were able to gain 4% of throughput compared to what the SPE has to offer. In the automobile industry context, this increase in throughput adds around $800k$ additional vehicles to the $20M$ already existing or could only use the saved resources in other matters that interest the Group differently. As future works, we are interested in reviewing the current Lambda architecture that proves to be limited in specific scenarios, notably the fault tolerance aspect of it. Another alternative that was proposed by the CEO of Confluent, Jay Kreps, the Kappa architecture, could be explored [110]. The idea is to use one stream processing engine for both batch and stream jobs accompanied by a log-persistent messaging system such as Kafka. Moreover, we plan on devising an algorithm that would consider developers'

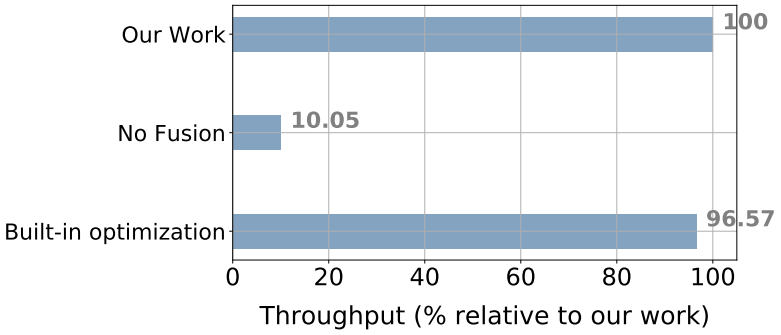


Figure 6.10: Eco-Driving scoring application performance

inputs as descriptive data for the application. Then, the algorithm with all the information provided will propose an efficient strategy scheme for deploying such applications.

IV

CONCLUSION

7

GENERAL CONCLUSION

7.1/ SUMMARY OF THE PHD THESIS

In this thesis, resource allocation optimization has been studied to ensure flawless, fast, and efficient data processing in stream processing engines and a proposal of an all-in-all architecture devised ideally for automotive Big Data applications. This dissertation comprises two parts: the first part covers the scientific and technological background of collecting and processing Big Data in the automotive context, whereas the second one presents the contributions made in this thesis.

The first part began by presenting the main concepts and technologies that led to smarter and more secure driving experience and allowed researchers to unlock new scientific challenges. For that reason, a history of Intelligent Transportation System was provided, then focused on one of the pillars of ITS, the Vehicular Network (VANET), which itself evolved into becoming the Internet of Vehicles (IoV). That enhancement made the cars connected to the grid and created a new challenge in the domain of data collection and analysis.
Second, we speak about the developed technology to deal precisely with the kind of scenarios developed in the first part. Hadoop Ecosystem or general-processing platform was made to deal with numerous big data applications in the Internet of Things (IoT). Hadoop platform allowed the agenda of ITS to move forward by implementing those massive computing platforms to deal with the vast amount of data being generated by millions of cars and sensors on the road.
Third, and after illustrating the technologies made for generally dealing with data-intensive data, we focus on one of the processing platform axes, the streaming processing. This chapter went through the history of stream processing engines and then described what this system is composed of. We listed then the challenges that can appear in the domain. Finally, some academic and commercial engines were surveyed.

The second part of this dissertation presented the contributions. The first research focused on a basic and essential task in Stream Processing Engine, a resource alloca-

tion algorithm. We tackled a central challenge in this work when deploying streaming applications, that is the process to map operators of a streaming application to available physical resources to enhance the application's performance. We addressed this problem by demonstrating that the inherent data parallelism approach could not be considered in all application scenarios as the generic solution. We targeted this issue by showing that the approach based on inherent data parallelism does not necessarily lead to the best performance for all applications. In fact, through a real-world application (i.e., the Eco-Driving service), we showed that a mapping based on insightful analysis of the target application's specifics and the infrastructure's features could significantly improve the application performance, leading thus, to a noticeable impact. The experiments backed by real data showed that our proposal improves over the straightforward approach by about 4% in terms of throughput. This enhancement allowed PSA's infrastructure to handle nearly 800 thousand additional vehicles over the expected 20 million CVs by 2025.

In the second research, we conceived a novel end-to-end architecture that answers data-intensive applications' requirements. Today, connected vehicles (CVs) can collect up to 170 different information (speed, temperature, fuel consumption, etc.) from onboard built-in sensors and transmit them, in a real-time fashion, to an infrastructure, usually by 4G/5G wireless communications. This reality brings with it many opportunities for developing new and innovative telematics services, including driver safety, customer experience, infotainment, quality and reliability, dealer services, location-based services, etc. Approximately 2 billion connected cars are projected to be on the world's roadways by the end of 2025, where each of them will produce up to 30 terabytes of data every day. Managing this Big Data in individual or batch mode imposes tight restrictions on the data processing software underlying it. In this chapter, we discussed precisely the architecture deployed by Groupe PSA on real CVs big data platform. Moreover, we presented technologies and open-source products used within different platform components to gather, store, process, and, more importantly, leverage big data.

We believe that the solutions proposed in this thesis for various challenges in the phases of stream processing applications and analysis in an IoV-scenario application can serve as additional and efficient solution choices or tools to address the challenges mentioned above in the automotive industries.

7.2/ PERSPECTIVES

We are witnessing a significant change in how we observe and understand the world around us. The big data phenomena challenged existing scientific methodologies and techniques and invited researchers to review how they work. The first paradigm of research methodology was initially mainly focused on experiments. The second paradigm

of theoretical science was focused primarily on the study of different theorems and laws. In reality, however, the theoretical analysis turned out to be too abstract and hard to implement in several situations for dealing with different practical issues. Researchers then began using simulation-based approaches that contributed to the third paradigm of computational science. In 2007, Jim Gray distinguished computational science from data-intensive research. Gray claimed that the fourth paradigm is not only a revolution in scientific research but also a revolution in people's thinking [111].
Likewise, big data analytics is becoming the backbone of every 21st-century enterprise. For example, in the business domain, opportunities to use big data tools to raise sales, cut costs, and manage risks are a representative sample of a long list of useful applications that will continue to evolve and develop. In particular, the lag in the use of big data technologies and applications has become the leading factor in the loss of strategic business advantages. Thus, any enterprise's ability to collect, store, process, and analyze vast data quantities would be a new landmark indicator for its intensity, capacity, and potential growth. Groupe PSA has also been subject to that revolution and decided to grasp the momentum to move with the trend. Companies like Groupe PSA understood this trend and invested heavily in the infrastructure, and they now contribute to the advancement of the field by discovering new potential scenarios that can profit from this new research paradigm to give value for the companies.
Despite the high expectations of the big data paradigm's promises and potentials, there are still many obstacles to harnessing its full strength [112]. In this thesis, we tackled the physical resource management problem. However, the management problem can have other faces and on other levels. We have seen a rise in the many types of data and the complexity of it. Second, we are witnessing a growth in the number of proposed large-scale processing platforms, making inter-platform compatibility a primordial task. Therefore, to help make platforms interoperable, the future focus should be invested in studying and benchmarking large-scale processing systems to propose an independent data scientist platform.

In a potential future work, the experience shared on the big data architecture in the automotive context can be developed to produce an all-inclusive architecture allowing many platforms to coexist. Practically, there is not so far a single platform that can, in any application scenario, outperform all other systems for different workloads. Moreover, moving data from one system to another for analytics jobs is supposedly more efficient and reliable and is a laborious and time-consuming task for engineers. As a result, users are obliged to focus on one platform even though it will not provide the best performance. Lastly, a research effort can be put in converging the world of Big Data analytics that masters the art of getting insights out of data in offline and online mode, with the world of High-Performance Computing (HPC) that can perform quadrillions of calculations per second. This marriage between the two could allow technologies such as the Internet of

Things (IoT), Artificial Intelligence (AI), and 3D imaging that produces data exponentially, to take benefit of the lightning-fast, highly reliable IT infrastructure to process, store, and analyze massive amounts of data.

BIBLIOGRAPHY

[1] TAL, I., AND MUNTEAN, G.-M. **Clustering and 5g-enabled smart cities: A survey of clustering schemes in vanets**. In *Paving the Way for 5G Through the Convergence of Wireless Systems*. IGI Global, 2019, pp. 18–55.

[2] CONTRERAS-CASTILLO, J., ZEADALLY, S., AND GUERRERO-IBAÑEZ, J. A. **Internet of vehicles: architecture, protocols, and security**. *IEEE internet of things Journal 5*, 5 (2017), 3701–3709.

[3] YANG, F., WANG, S., LI, J., LIU, Z., AND SUN, Q. **An overview of internet of vehicles**. *China communications 11*, 10 (2014), 1–15.

[4] LEE, E.-K., GERLA, M., PAU, G., LEE, U., AND LIM, J.-H. **Internet of vehicles: From intelligent grid to autonomous cars and vehicular fogs**. *International Journal of Distributed Sensor Networks 12*, 9 (2016), 1550147716665500.

[5] KAIWARTYA, O., ABDULLAH, A. H., CAO, Y., ALTAMEEM, A., PRASAD, M., LIN, C.-T., AND LIU, X. **Internet of vehicles: Motivation, layered architecture, network model, challenges, and future aspects**. *IEEE Access 4* (2016), 5356–5373.

[6] NITTI, M., GIRAU, R., FLORIS, A., AND ATZORI, L. **On adding the social dimension to the internet of vehicles: Friendship and middleware**. In *2014 IEEE international black sea conference on communications and networking (BlackSeaCom)* (2014), IEEE, pp. 134–138.

[7] GUO, L., DONG, M., OTA, K., LI, Q., YE, T., WU, J., AND LI, J. **A secure mechanism for big data collection in large scale internet of vehicle**. *IEEE Internet of Things Journal 4*, 2 (2017), 601–610.

[8] DARWISH, T. S., AND BAKAR, K. A. **Fog based intelligent transportation big data analytics in the internet of vehicles environment: motivations, architecture, challenges, and critical issues**. *IEEE Access 6* (2018), 15679–15701.

[9] GOLESTAN, K., SOUA, R., KARRAY, F., AND KAMEL, M. S. **Situation awareness within the context of connected cars: A comprehensive review and recent trends**. *Information Fusion 29* (2016), 68–83.

[10] YANG, F., LI, J., LEI, T., AND WANG, S. **Architecture and key technologies for internet of vehicles: a survey**.

[11] CONTRERAS-CASTILLO, J., ZEADALLY, S., AND GUERRERO IBÁÑEZ, J. A. **A seven-layered model architecture for internet of vehicles**. *Journal of Information and Telecommunication 1*, 1 (2017), 4–22.

[12] SHERAZI, H. H. R., KHAN, Z. A., IQBAL, R., RIZWAN, S., IMRAN, M. A., AND AWAN, K. **A heterogeneous iov architecture for data forwarding in vehicle to infrastructure communication**. *Mobile Information Systems 2019* (2019).

[13] FIGUEIREDO, L., JESUS, I., MACHADO, J. T., FERREIRA, J. R., AND DE CARVALHO, J. M. **Towards the development of intelligent transportation systems**. In *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585)* (2001), IEEE, pp. 1206–1211.

[14] MASAKI, I. **Machine-vision systems for intelligent transportation systems**. *IEEE Intelligent Systems and their applications 13*, 6 (1998), 24–31.

[15] ROBERT, L. **French, r&d french associates,"the ieee and its"**. *IEEE Intelligent Systems, pag* (1999), 75–77.

[16] GRAEFE, V., AND KUHNERT, K.-D. **Vision-based autonomous road vehicles**. In *Vision-based vehicle guidance*. Springer, 1992, pp. 1–29.

[17] ULMER, B. **Vita ii-active collision avoidance in real traffic**. In *Proceedings of the Intelligent Vehicles' 94 Symposium* (1994), IEEE, pp. 1–6.

[18] TOMIZUKA, M. **Automated highway systems-an intelligent transportation system for the next century**. In *ISIE'97 Proceeding of the IEEE International Symposium on Industrial Electronics* (1997), vol. 1, IEEE, pp. PS1–PS4.

[19] BETSOLD, R. **Intelligent vehicle/highway systems for the united states-an emerging national program**. In *JSK International Symposium-Technological Innovations for Tomorrow's Automobile Traffic and Driving Information Systems* (1989), pp. 53–59.

[20] AMERICA, I. **Strategic plan for ivhs in the united states**. *Washington, DC: Author* (1992).

[21] NAKASHITA, H., OKAMOTO, H., AND KAWABATA, T. **Advanced mobile traffic information and communication system (amtics)**. In *Transportation Research Board Meeting* (1988).

[22] TOKUYAMA, H. **Asia-pacific projects status and plans**. *US Dept. of Transportation* (1997).

[23] SHAWE-TAYLOR, J., DE BIE, T., AND CRISTIANINI, N. **Data mining, data fusion and information management**. In *IEE Proceedings-Intelligent Transport Systems* (2006), vol. 153, IET, pp. 221–229.

[24] PETERS, A., VON KLOT, S., HEIER, M., TRENTINAGLIA, I., HÖRMANN, A., WICHMANN, H. E., AND LÖWEL, H. **Exposure to traffic and the onset of myocardial infarction**. *New England Journal of Medicine 351*, 17 (2004), 1721–1730.

[25] ZHENG, N.-N., TANG, S., CHENG, H., LI, Q., LAI, G., AND WANG, F.-W. **Toward intelligent driver-assistance and safety warning system**. *IEEE Intelligent Systems 19*, 2 (2004), 8–11.

[26] MALTA, L., MIYAJIMA, C., AND TAKEDA, K. **A study of driver behavior under potential threats in vehicle traffic**. *IEEE Transactions on Intelligent Transportation Systems 10*, 2 (2009), 201–210.

[27] **Traffic incident management handbook**. https://rosap.ntl.bts.gov/view/dot/3702, Nov. 2000. Accessed: 2020-08-21.

[28] CHIU, Y.-C., AND MIRCHANDANI, P. **Online behavior-robust feedback information routing strategy for mass evacuation**. *Intelligent Transportation Systems, IEEE Transactions on 9* (07 2008), doi:10.1109/TITS.2008.922878, 264 – 274.

[29] ZENG, D. D., CHAWATHE, S., HUANG, H., AND WANG, F. **Protecting transportation infrastructure**. *IEEE Intelligent Systems 22* (10 2007), doi:10.1109/MIS.2007.4338487, 8 – 11.

[30] HAMZA-LUP, G., HUA, K., LE, M., AND PENG, R. **Dynamic plan generation and real-time management techniques for traffic evacuation**. *Intelligent Transportation Systems, IEEE Transactions on 9* (01 2009), doi:10.1109/TITS.2008.2006738, 615 – 624.

[31] SUSSMAN, J. **Perspectives on intelligent transportation systems (its)**. doi:10.1007/b101063, *Perspectives on Intelligent Transportation Systems (ITS), by J.M. Sussman. X, 229 p. 0-387-23257-5. Berlin: Springer, 2005.* (01 2005).

[32] DEAN, J., AND GHEMAWAT, S. **Mapreduce: Simplified data processing on large clusters**.

[33] WHITE, T. **Hadoop: The definitive guide**. " O'Reilly Media, Inc.", 2012.

[34] BU, Y., HOWE, B., BALAZINSKA, M., AND ERNST, M. D. **Haloop: Efficient iterative data processing on large clusters**. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 285–296.

[35] YANG, H.-C., DASDAN, A., HSIAO, R.-L., AND PARKER, D. S. **Map-reduce-merge: simplified relational data processing on large clusters**. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (2007), pp. 1029–1040.

[36] NYKIEL, T., POTAMIAS, M., MISHRA, C., KOLLIOS, G., AND KOUDAS, N. **Mrshare: sharing across multiple queries in mapreduce**. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 494–505.

[37] ELTABAKH, M. Y., TIAN, Y., ÖZCAN, F., GEMULLA, R., KRETTEK, A., AND MCPHERSON, J. **Cohadoop: flexible data placement and its exploitation in hadoop**. *Proceedings of the VLDB Endowment 4*, 9 (2011), 575–585.

[38] HUAI, Y., CHAUHAN, A., GATES, A., HAGLEITNER, G., HANSON, E. N., O'MALLEY, O., PANDEY, J., YUAN, Y., LEE, R., AND ZHANG, X. **Major technical advancements in apache hive**. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), pp. 1235–1246.

[39] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. **Pregel: a system for large-scale graph processing**. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), pp. 135–146.

[40] ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., SHENKER, S., STOICA, I., AND OTHERS. **Spark: Cluster computing with working sets.** *HotCloud 10*, 10-10 (2010), 95.

[41] ODERSKY, M., SPOON, L., AND VENNERS, B. **Programming in scala: A comprehensive step-by-step guide, 2nd**. *USA: Artima Incorporation* (2011).

[42] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. **The hadoop distributed file system**. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (2010), Ieee, pp. 1–10.

[43] ARMBRUST, M., XIN, R. S., LIAN, C., HUAI, Y., LIU, D., BRADLEY, J. K., MENG, X., KAFTAN, T., FRANKLIN, M. J., GHODSI, A., AND OTHERS. **Spark sql: Relational data processing in spark**. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (2015), pp. 1383–1394.

[44] GONZALEZ, J. E., XIN, R. S., DAVE, A., CRANKSHAW, D., FRANKLIN, M. J., AND STOICA, I. **Graphx: Graph processing in a distributed dataflow framework**. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)* (2014), pp. 599–613.

[45] SPARKS, E. R., TALWALKAR, A., SMITH, V., KOTTALAM, J., PAN, X., GONZALEZ, J., FRANKLIN, M. J., JORDAN, M. I., AND KRASKA, T. **Mli: An api for distributed machine learning**. In *2013 IEEE 13th International Conference on Data Mining* (2013), IEEE, pp. 1187–1192.

[46] NARKHEDE, N., SHAPIRA, G., AND PALINO, T. **Kafka: the definitive guide: real-time data and stream processing at scale**. " O'Reilly Media, Inc.", 2017.

[47] CONDIE, T., CONWAY, N., ALVARO, P., HELLERSTEIN, J. M., ELMELEEGY, K., AND SEARS, R. **Mapreduce online.** In *Nsdi* (2010), vol. 10, p. 20.

[48] BHATOTIA, P., WIEDER, A., RODRIGUES, R., ACAR, U. A., AND PASQUIN, R. **Incoop: Mapreduce for incremental computations**. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), pp. 1–14.

[49] SAKR, S. **Big Data 2.0 Processing Systems: A Systems Overview**. Springer Nature, 2020.

[50] CLIFFORD, S., AND HARDY, Q. **Attention, shoppers: Store is tracking your cell**, 2013.

[51] HA, K., CHEN, Z., HU, W., RICHTER, W., PILLAI, P., AND SATYANARAYANAN, M. **Towards wearable cognitive assistance**. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services* (2014), pp. 68–81.

[52] ASSUNÇÃO, M. D., CALHEIROS, R. N., BIANCHI, S., NETTO, M. A., AND BUYYA, R. **Big data computing and clouds: Trends and future directions**. *Journal of Parallel and Distributed Computing 79* (2015), 3–15.

[53] ATZORI, L., IERA, A., AND MORABITO, G. **The internet of things: A survey**. *Computer networks 54*, 15 (2010), 2787–2805.

[54] RETTIG, L., KHAYATI, M., CUDRÉ-MAUROUX, P., AND PIÓRKOWSKI, M. **Online anomaly detection over big data streams**. In *Applied Data Science*. Springer, 2019, pp. 289–312.

[55] ELLIS, B. **Real-time analytics: Techniques to analyze and visualize streaming data**. John Wiley & Sons, 2014.

[56] ALLEN, S. T., JANKOWSKI, M., AND PATHIRANA, P. **Storm Applied: Strategies for real-time event processing**. Manning Publications Co., 2015.

[57] BUYYA, R., AND DASTJERDI, A. V. **Internet of Things: Principles and paradigms**. Elsevier, 2016.

[58] DE ASSUNCAO, M. D., DA SILVA VEITH, A., AND BUYYA, R. **Distributed data stream processing and edge computing: A survey on resource elasticity and future directions**. *Journal of Network and Computer Applications 103* (2018), 1–17.

[59] CENTENARO, M., VANGELISTA, L., ZANELLA, A., AND ZORZI, M. **Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios**. *IEEE Wireless Communications 23*, 5 (2016), 60–67.

[60] DEAN, J., AND GHEMAWAT, S. **Mapreduce: simplified data processing on large clusters**. *Communications of the ACM 51*, 1 (2008), doi:10.1145/1327452.1327492, 107–113.

[61] BORTHAKUR, D., GRAY, J., SARMA, J. S., MUTHUKKARUPPAN, K., SPIEGELBERG, N., KUANG, H., RANGANATHAN, K., MOLKOV, D., MENON, A., RASH, S., AND OTHERS. **Apache hadoop goes realtime at facebook**. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (2011), pp. 1071–1080.

[62] CHEN, Y., ALSPAUGH, S., BORTHAKUR, D., AND KATZ, R. **Energy efficiency for large-scale mapreduce workloads with significant interactive analysis**. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), pp. 43–56.

[63] HAN, J., HAIHONG, E., LE, G., AND DU, J. **Survey on nosql database**. In *2011 6th international conference on pervasive computing and applications* (2011), IEEE, pp. 363–366.

[64] HIRZEL, M., SOULÉ, R., SCHNEIDER, S., GEDIK, B., AND GRIMM, R. **A catalog of stream processing optimizations**. *ACM Comput. Surv. 46*, 4 (Mar. 2014), doi:10.1145/2528412, 46:1–46:34.

[65] ANDRADE, H., GEDIK, B., AND TURAGA, D. **Fundamentals of Stream Processing: Application Design, Systems, and Analytics**. Cambridge University Press, 2014.

[66] FIGUEIRAS, P., HERGA, Z., GUERREIRO, G., ROSA, A., COSTA, R., AND JARDIM-GONÇALVES, R. **Real-time monitoring of road traffic using data stream mining**. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)* (2018), IEEE, pp. 1–8.

[67] AMARASINGHAM, R., PRONOVOST, P. J., DIENER-WEST, M., GOESCHEL, C., DORMAN, T., THIEMANN, D. R., AND POWE, N. R. **Measuring clinical information technology in the icu setting: application in a quality improvement col-**

laborative. *Journal of the American Medical Informatics Association 14*, 3 (2007), 288–294.

[68] BLOUNT, M., EBLING, M. R., EKLUND, J. M., JAMES, A. G., MCGREGOR, C., PERCIVAL, N., SMITH, K., AND SOW, D. **Real-time analysis for intensive care: development and deployment of the artemis analytic system**. *IEEE Engineering in Medicine and Biology Magazine 29*, 2 (2010), 110–118.

[69] CUGOLA, G., AND MARGARA, A. **Processing flows of information: From data stream to complex event processing**. *ACM Computing Surveys (CSUR) 44*, 3 (2012), 1–62.

[70] CLINGER, W. D. **Foundations of actor semantics**. *AITR-633* (1981).

[71] **RDMA Consortium**. http://www.rdmaconsortium.org/. Accessed: 2020-08-20.

[72] SCHNEIDER, S., HIRZEL, M., AND GEDIK, B. **Tutorial: stream processing optimizations**. In *Proceedings of the 7th ACM international conference on Distributed event-based systems* (2013), pp. 249–258.

[73] HAROUN, A., MOSTEFAOUI, A., AND DESSABLES, F. **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)* (2017), IEEE, pp. 921–928.

[74] MARZ, N., AND WARREN, J. **Big Data: Principles and Best Practices of Scalable Realtime Data Systems**, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2015.

[75] DURI, S., ELLIOTT, J., GRUTESER, M., LIU, X., MOSKOWITZ, P., PEREZ, R., SINGH, M., AND TANG, J.-M. **Data protection and data sharing in telematics**. *Mob. Netw. Appl. 9*, 6 (Dec. 2004), 693–701.

[76] JOHANSON, M., BELENKI, S., JALMINGER, J., FANT, M., AND GJERTZ, M. **Big automotive data: Leveraging large volumes of data for knowledge-driven product development**. In *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014* (2014), doi:10.1109/BigData.2014.7004298, pp. 736–741.

[77] LI, Q., WANG, Z.-Y., LI, W.-H., LI, J., WANG, C., AND DU, R.-Y. **Applications integration in a hybrid cloud computing environment: Modelling and platform**. *Enterprise Information Systems 7*, 3 (2013), 237–271.

[78] BROUSMICHE, K. L., HENO, T., POULAIN, C., DALMIERES, A., AND HAMIDA, E. B. **Digitizing, securing and sharing vehicles life-cycle over a consortium**

blockchain: Lessons learned. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)* (2018), IEEE, pp. 1–5.

[79] BROUSMICHE, K. L., DURAND, A., HENO, T., POULAIN, C., DALMIERES, A., AND HAMIDA, E. B. **Hybrid cryptographic protocol for secure vehicle data sharing over a consortium blockchain**. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (2018), IEEE, pp. 1281–1286.

[80] DAVENPORT, T. H., BARTH, P., AND BEAN, R. **How big data is different**. *MIT Sloan Management Review 54*, 1 (2012), 43.

[81] DI NATALE, M., ZENG, H., GIUSTO, P., AND GHOSAL, A. **Understanding and using the controller area network communication protocol: theory and practice**. Springer Science & Business Media, 2012.

[82] ISO. **Road vehicles - controller area network (can) - part 1: Data link layer and physical signalling**. https://www.iso.org/standard/63648.html, November 2019.

[83] APACHE. **Kafka**. https://kafka.apache.org/.

[84] BALALAIE, A., HEYDARNOORI, A., AND JAMSHIDI, P. **Microservices architecture enables devops: Migration to a cloud-native architecture**. *IEEE Software 33*, 3 (May 2016), 42–52.

[85] HIRZEL, M., ANDRADE, H., GEDIK, B., JACQUES-SILVA, G., KHANDEKAR, R., KUMAR, V., MENDELL, M., NASGAARD, H., SCHNEIDER, S., SOULÉ, R., AND WU, K.-L. **Ibm streams processing language: Analyzing big data in motion**. *IBM J. Res. Dev. 57*, 3-4 (May 2013), doi:10.1147/JRD.2013.2243535, 1:7–1:7.

[86] APACHE. **Samza**. http://samza.apache.org/.

[87] DIXON, J. **Pentaho, hadoop, and data lakes**. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

[88] FANG, H. **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)* (2015), IEEE, pp. 820–824.

[89] APACHE. **Hadoop**. https://hadoop.apache.org/. version 2.7.3.

[90] ACHARYA, S., RAJESH, B., VAGHANI, S. B., SOKOLINSKI, I., CHIAO-CHUAN, S., AND DESAI, K. **Object storage system**, July 8 2014. US Patent 8,775,773.

[91] DEAN, J., AND GHEMAWAT, S. **Mapreduce: A flexible data processing tool**. *Commun. ACM 53*, 1 (Jan. 2010), 72–77.

[92] APACHE. **Spark**. http://spark.apache.org/. version 1.6.0.

[93] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. **Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing**. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), USENIX Association, pp. 2–2.

[94] VAN DER LANS, R. **Data Virtualization for business intelligence systems: revolutionizing data integration for data warehouses**. Elsevier, 2012.

[95] HAN, J., E, H., LE, G., AND DU, J. **Survey on nosql database**. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on* (Oct 2011), doi:10.1109/ICPCA.2011.6106531, pp. 363–366.

[96] APACHE. **Hbase**. http://hbase.apache.org/.

[97] APACHE. **Phoenix**. https://phoenix.apache.org/.

[98] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. **Bigtable: A distributed storage system for structured data**. *ACM Trans. Comput. Syst. 26*, 2 (June 2008), 4:1–4:26.

[99] STROZZI, C. **Nosql: a non-sql rdbms**, 1998.

[100] ADAMCZYK, P., SMITH, P. H., JOHNSON, R. E., AND HAFIZ, M. **Rest and web services: In theory and in practice**. In *REST: from research to practice*. Springer, 2011, pp. 35–57.

[101] FIELDING, R. T., AND TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**, vol. 7. University of California, Irvine Irvine, 2000.

[102] FERREIRA, J. C., DE ALMEIDA, J., AND DA SILVA, A. R. **The impact of driving styles on fuel consumption: A data-warehouse-and-data-mining-based discovery process**. *IEEE Transactions on Intelligent Transportation Systems 16*, 5 (Oct 2015), 2653–2662.

[103] SERHANI, M. A., EL KASSABI, H. T., TALEB, I., AND NUJUM, A. **An hybrid approach to quality evaluation across big data value chain**. In *2016 IEEE International Congress on Big Data (BigData Congress)* (2016), IEEE, pp. 418–425.

[104] LEE, Y. W., STRONG, D. M., KAHN, B. K., AND WANG, R. Y. **Aimq: a methodology for information quality assessment**. *Information & management 40*, 2 (2002), 133–146.

[105] HAZEN, B. T., BOONE, C. A., EZELL, J. D., AND JONES-FARMER, L. A. **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**. *International Journal of Production Economics 154* (2014), 72–80.

[106] BATINI, C., CAPPIELLO, C., FRANCALANCI, C., AND MAURINO, A. **Methodologies for data quality assessment and improvement**. *ACM computing surveys (CSUR) 41*, 3 (2009), 1–52.

[107] CHOUALI, S., BOUKERCHE, A., AND MOSTEFAOUI, A. **Towards a formal analysis of mqtt protocol in the context of communicating vehicles**. In *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access, MOBIWAC 2017, Miami, FL, USA, November 21 - 25, 2017* (2017), ACM International Symposium on Mobility Management and Wireless Access, doi:10.1145/3132062.3132079, pp. 129–136.

[108] **IBM Streams**, Feb 2018.

[109] GEDIK, B., ANDRADE, H., AND WU, K.-L. **A Code Generation Approach to Optimizing High-performance Distributed Data Stream Processing**. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (New York, NY, USA, 2009), CIKM '09, ACM, doi:10.1145/1645953.1646061, pp. 847–856.

[110] KREPS, J. **Questioning the Lambda Architecture**, Jul 2014.

[111] HEY, T., TANSLEY, S., AND TOLLE, K. **The fourth paradigm: Data-intensive scientific discovery. microsoft research, redmond, washington (2009)**. *URL http://research. microsoft. com/en-us/collaboration/fourthparadigm* (2009).

[112] JAGADISH, H. V., GEHRKE, J., LABRINIDIS, A., PAPAKONSTANTINOU, Y., PATEL, J. M., RAMAKRISHNAN, R., AND SHAHABI, C. **Big data and its technical challenges**. *Communications of the ACM 57*, 7 (2014), 86–94.

List of Figures

List of Tables

V

APPENDIX

A

Appendix: Presentation of Groupe PSA

A.1/ Presentation of Groupe PSA

The PSA Group, formerly called PSA Peugeot Citroën, is a French automobile manufacturer, created in 1810.

A.1.1/ Market and activities

A./ Economic situation

The PSA Group was, in 2018, the second European car manufacturer, with a turnover of 74 billion euros.

The creation and sale of cars is its core business, and with its three main brands, Peugeot, Citroën and DS, more than 3.9 million cars were sold in 2017.

With more than 170,000 points of sale throughout the world, its main sales market is Europe, but it is also very present on other continents, being the number 1 in China, for example.

Groupe PSA also has industrial sites all over the world, as seen in Figure A.1.

To follow a period of crisis, between 2012 and 2014, the group decided to launch a new plan, called Push to Pass, which consists of reducing the various expenses within the company, without compromising production, in order to provide a higher operating margin, which returned to positive in 2015, after 3 years of recession.

This plan can be summarized in five phases for the company:

- *From the product to the customer*, trying to stand out from its competitors by con-

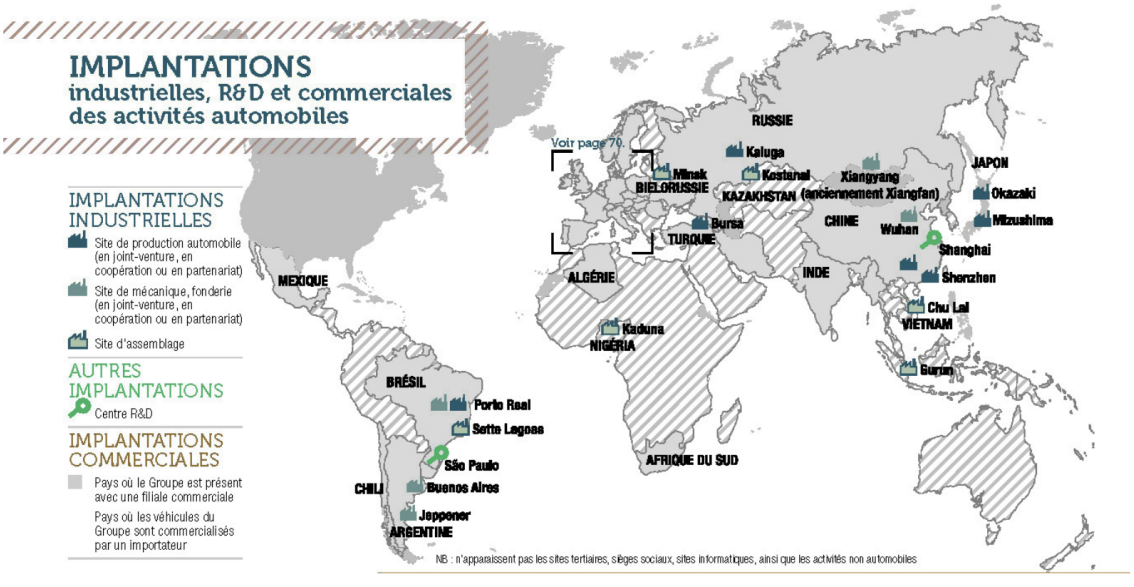


Figure A.1: Establishment of Groupe PSA industrial sites around the world

tinuing to divide its different ranges of vehicles according to its brands.

- *From ownership to experience*, giving more character to your cars, so that buying a car is no longer just out of necessity, but also out of pleasure.
- *From car to mobility*, continuing to research in various areas of innovation, such as autonomous driving.
- *From a single activity to a portfolio of activities*, by diversifying its activity, because even if its core business remains the automotive sector, new opportunities are available to it, such as the sale of data collected by its cars.
- *From local to global*, continuing to make progress in markets abroad, particularly in Asia, which is experiencing strong competition in the automotive sector.

B./ ACTIVITIES

The profits of this company are generated mainly from the sale of cars of its three main brands (Peugeot, Citroën and DS). By providing a clear demarcation for the public of each of these brands, the group succeeds in being the 7th largest car manufacturer in the world.

In addition to selling cars, the group has commercial activities in two other sectors:

- The **automotive equipment sector**, through the Faurecia group, which produces automotive equipment, such as seats, windshield wipers, bumpers, etc. for all car

brands.

- The **financial sector**, through Banque PSA Finance, which allows the group's car buyers to obtain financing at lower rates when purchasing their vehicles. The latter, which has the status of a financial institution, will soon be joined (probably merged) by the GM Europe bank, which the PSA Group acquired at the same time as the Opel and Vauxhall brands.

A.1.2/ Main brands

A./ Peugeot

The Peugeot brand, created in 1810, is now present in more than 160 countries, through more than 10,000 points of sale. It aims to be a generalist medium / high-end brand with a global vocation, combining Requirement, Allure and Emotion.

In 2018, its sales increased by 6.2%, from 1,312,082 to 1,393,431 vehicles sold worldwide.

B./ Citroën

The Citroën company, founded in 1919, was bought by Peugeot in 1976. It aims to be the group's mid-range brand, so as not to compete directly with Peugeot or DS.

It saw its sales increase by 6.2% in 2018, going from 1,171,746 to 1,244,394 units sold.

C./ DS Automobiles

The DS brand, created in 2009, was initially a line within the Citroën brand, but stood out from it in June 2014, by becoming a brand, to give these different lines of cars two distinct identities.

By being the group's top-of-the-range brand, and by offering a personalization of each of its models, it attracts a small audience, mainly on the European market.

Its sales increased by 12.6%, in 2018, from 74,201 to 83,550 vehicles sold.

D./ Opel

This German automobile brand, created in 1862, was bought in March 2017 by Groupe PSA for the sum of $ 1.4 billion.

Initially, the PSA Group will focus on helping Opel to fill its deficit, so that this brand can subsequently become another of the main brands of the group.

Its sales are down slightly, by -1.3% in 2018.

E./ VAUXHALL

This English car brand, created in 1857, was acquired in March 2017 at the same time as the Opel brand; both previously belonging to the GM Europe group.

In the same way as for Opel, the PSA Group will try to adapt the company's processes to those of the group, gradually, while providing it with financial support.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

école doctorale **sciences pour l'ingénieur et microtechniques**

Title: Optimizing Resource Utilization in Distributed Computing Systems for Automotive Applications

Keywords: Big Data Architecture, Query Optimization, Cloud Computing, Hadoop Ecosystem, Stream Computing, Connected Vehicles

Abstract:

One of the main challenges for the automobile industry in the digital age is to provide their customers with a reliable and ubiquitous level of connected services. Smart cars have been entering the market for a few years now to offer drivers and passengers safer, more comfortable, and entertaining journeys. All this by designing, behind the scenes, computer systems that perform well while conserving the use of resources.
The performance of a Big Data architecture in the automotive industry relies on keeping up with the growing trend of connected vehicles and maintaining a high quality of service. The Cloud at Groupe PSA has a particular load on ensuring a real-time data processing service for all the brand's connected vehicles: with 200k connected vehicles sold each year, the infrastructure is continuously challenged.
Therefore, this thesis mainly focuses on optimizing resource allocation while considering the specifics of continuous flow processing applications and proposing a modular and fine-tuned component architecture for automotive scenarios.

First, we go over a fundamental and essential process in Stream Processing Engines, a resource allocation algorithm. The central challenge of deploying streaming applications is mapping the operator graph, representing the application logic, to the available physical resources to improve its performance. We have targeted this problem by showing that the approach based on inherent data parallelism does not necessarily lead to all applications' best performance.
Second, we revisit the Big Data architecture and design an end-to-end architecture that meets today's demands of data-intensive applications. We report on CV's Big Data platform, particularly the one deployed by Groupe PSA. In particular, we present open source technologies and products used in different platform components to collect, store, process, and, most importantly, exploit big data and highlight why the Hadoop system is no longer the de-facto solution of Big Data. We end with a detailed assessment of the architecture while justifying the choices made during design and implementation.

Titre : Optimisation de l'utilisation des ressources dans les systèmes informatiques distribués pour les applications automobiles

Mots-clés : architecture Big Data, optimisation de requêtes, Cloud Computing, écosystème Hadoop, Stream Computing, véhicules connectés

Résumé :

L'un des principaux défis de l'industrie automobile à l'ère du numérique est de fournir à ses clients un niveau fiable et omniprésent de services connectés. Les voitures intelligentes font leur entrée sur le marché depuis quelques années maintenant pour offrir aux conducteurs et aux passagers des trajets plus sûrs, plus confortables et plus divertissants. Tout cela en concevant, dans les coulisses, des systèmes informatiques performants tout en préservant l'utilisation des ressources.
La performance d'une architecture Big Data dans l'industrie automobile repose sur le maintien de la tendance croissante des véhicules connectés et le maintien d'une qualité de service élevée. Le Cloud du Groupe PSA a un souci particulier d'assurer un service de traitement des données en temps réel pour tous les véhicules connectés de la marque: avec 200k véhicules connectés vendus chaque année, l'infrastructure est constamment remise en question.
Par conséquent, cette thèse se concentre principalement sur l'optimisation de l'allocation des ressources tout en considérant les spécificités des applications de traitement en flux continu et en proposant une architecture de composants modulaire et affinée pour les scénarios automobiles.

Premièrement, nous passons en revue un processus fondamental et essentiel dans Stream Processing Engines, un algorithme d'allocation de ressources. Le défi central du déploiement d'applications de streaming consiste à mapper le graphe opérateur, représentant la logique de l'application, aux ressources physiques disponibles pour améliorer ses performances. Nous avons ciblé ce problème en montrant que l'approche basée sur le parallélisme des données inhérent ne conduit pas nécessairement aux meilleures performances de tous les types d'applications.
Deuxièmement, nous revisitons l'architecture Big Data et concevons une architecture de bout en bout qui répond aux exigences d'aujourd'hui des applications gourmandes en données. Nous faisons rapport de la plateforme Big Data de CV, notamment celle déployée par le Groupe PSA. En particulier, nous présentons des technologies et des produits open source utilisés dans différents composants de plate-forme pour collecter, stocker, traiter et, surtout, exploiter le Big Data et souligner pourquoi le système Hadoop n'est plus la solution de facto du Big Data. Nous terminons par un bilan détaillé de l'architecture tout en justifiant les choix effectués lors de la conception et de la mise en œuvre.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ
Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon, France