Open Journal of Applied Sciences, 2023, 13, 198-206
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Events Sourcing and Command Query Responsibility Segregation Based Fast Data Architecture

Gérard Behou N'guessan[1*], Odilon Yapo Achiepo[1], Jérôme Diako[2]

[1]Research and Digital Expertise Unit (UREN), Virtual University of Côte d'Ivoire (UVCI), Abidjan, Ivory Coast
[2]Lastic, ESATIC, Abidjan, Ivory Coast
Email: *behou.nguessan@uvci.edu.ci

How to cite this paper: N'guessan, G.B., Achiepo, O.Y. and Diako, J. (2023) Events Sourcing and Command Query Responsibility Segregation Based Fast Data Architecture. *Open Journal of Applied Sciences*, **13**, 198-206.
https://doi.org/10.4236/ojapps.2023.132016

Received: December 5, 2022
Accepted: February 14, 2023
Published: February 17, 2023

Abstract

With the advent of Big Data, the fields of Statistics and Computer Science coexist in current information systems. In addition to this, technological advances in embedded systems, in particular Internet of Things technologies, make it possible to develop real-time applications. These technological developments are disrupting Software Engineering because the use of large amounts of real-time data requires advanced thinking in terms of software architecture. The purpose of this article is to propose an architecture unifying not only Software Engineering and Big Data activities, but also batch and streaming architectures for the exploitation of massive data. This architecture has the advantage of making possible the development of applications and digital services exploiting very large volumes of data in real time; both for management needs and for analytical purposes. This architecture was tested on COVID-19 data as part of the development of an application for real-time monitoring of the evolution of the pandemic in Côte d'Ivoire using PostgreSQL, ELasticsearch, Kafka, Kafka Connect, NiFi, Spark, Node-Red and MoleculerJS to operationalize the architecture.

Keywords

Architecture, Software Engineering, Big Data, Data Engineering, Real Time

1. Introduction

The proliferation of data has caused a lot of concern for some companies since the advent of digital science. Those companies often need these data in the implementation of a decision-making strategy. It often happens that the data, because they originate from various sources, they represent a real challenge for the companies. In order to efficiently exploit the data with a very low latency time, certain real-time architectures have emerged. These architectures are much more used in the management of raw Big Data generated from various sources. These Big Data are characterized by their volume, variety and velocity; which means that their storage and operation require capacities that surpass those of traditional computer systems [1].

At the origin of big data, the lambda architecture was proposed for the work of integration and exploitation of massive data [2]. This architecture is still widely used in Big Data industrialization activities in companies and organizations. However, real-time application requirements, particularly with the advent of the Internet of Things, have led to the proposal of smack architectures as the first viable industrial approach [3]. Thus, the lambda and smack architectures are intended to be references in the development of applications exploiting Big Data in particular, data integration applications.

However, all these proposed so-called existing architectures proposed have the disadvantage of not taking into account application development activities as well as the deployment of Machine Learning solutions. They are therefore incomplete for the creation of an integrated and coherent enterprise architecture. This is why, in this paper, we propose a so-called Payi architecture that addresses all these limits. From a technical point of view, the Payi architecture is based on Command Query Responsibility Segregation (CQRS) and Event Sourcing approaches, integrating Software Engineering, Data Engineering and Machine Learning model deployment activities simultaneously.

The first is to present a literature review of CQRS and Event Sourcing approaches. Then, this will then lead to the presentation of the new proposed architecture. And finally we end with a conclusion.

2. Existing Architectures

2.1. Lambda Architecture

At the origin of the industrialization of Big Data, the lambda architecture was proposed. Figure 1 below gives the structure of the lambda architecture:

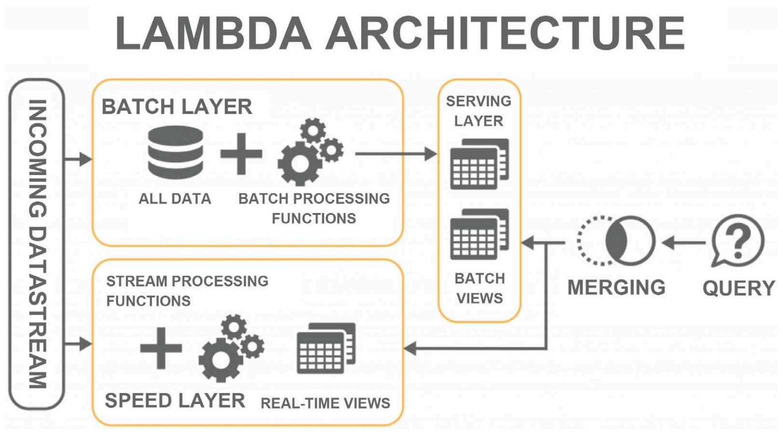

Figure 1. Lambda architecture (Source: [4]).

The lambda architecture is composed of four (4) parts namely the data source, the batch layer, the streaming layer and the data presentation layer.

The data source: this notion is used as a parameter for queries and concerns structured, unstructured and semi-structured data. It comes in various forms and makes it possible to determine the data to which the request must relate [5].

The batch layer: This layer has the role of storing the constantly growing and immutable basic data in a file system such as the HDFS. It also pre-calculates batch views of distributed data using the MapReduce function of this layer. Batch views are commonly used to respond to incoming requests with low read latency [6] [7].

The streaming layer: its role is to route the data to the data processor. Then, it processes them in real time and produces a result within a short time [8]. The output of this processing is passed to the service layer where this data is displayed to the end user as part of a web application, dashboard, report or event used by another system [9].

The presentation layer: it consists of frontend components as well as micro-services in order to serve the frontend. This layer is implemented in a pluggable manner so that the non-functional requirement of extensibility is satisfied [10].

The specificity of this basic architecture is that it separates batch data processing from real-time processing.

In the quest for better architecture, Miguel *et al.* proposed an improved version of the Lambda architecture. Their approach is to standardize and simplify data extraction. To do this, they propose to insert a data ingestion layer between the data source and the batch and real-time data integration layers [2]. In this same vision, Tarik Hachad *et al.* also proposed a new Big Data architecture, based on the lambda architecture, used in the deployment of Machine Learning models. In their architecture, the data source is replaced by the new data to be used for predictions thanks to an already trained machine learning model. Their proposal was used to detect the level of attention of students be it in a batch context as well as in a real time application [3].

2.2. The SMACK Architecture

In the professional context of Fast Data, an architecture based on Scala technologies such as Spark, Mesos, Akka, Cassandra and Kafka (SMACK architecture) has been very popular in companies. The following **Figure 2** shows the SMACK architecture:

The SMACK architecture is composed of 4 layers namely the data source, the data propagation layer (Kafka), the processing layer (Spark) and the data storage layer (Cassandra). These four (04) layers use two additional technologies to accelerate the distribution of real-time data. These are the Akka actor-model programming paradigm and the Mesos machine cluster management technology.

Spark: is a processing engine within the Big Data architecture. It performs

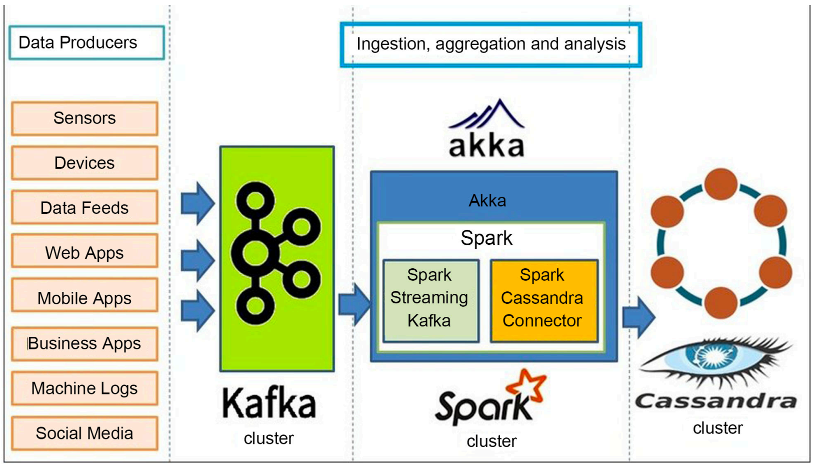

Figure 2. SMACK architecture (Source: [11]).

analytical work on real-time data. This engine offers flexibility from a development perspective and is available on scala, Java, R, Python and SQL. This engine offers the infrastructure and operation of the worker in a program [12]. It can also be used as a data science tool, capable of handling large datasets and performing operations on them. It works with resilient distributed datasets (RDD), which provides fault tolerance, efficiency, speed, and in-memory data storage [13]. RDDs enforce immutability and have no negative effects of interfering parallel running tasks. When he this one runs on a cluster, a Spark driver program delegates work as tasks to its subsidiary worker nodes. This allows for scalability and is perfect for working in the cloud environment. Spark can work with different types of cluster managers, but in the SMACK stack.

My bones: is used to manage and coordinate cluster resources. It extracts all the computing resources (CPU, memory, storage) from the different machines. It is not only the core of distributed systems, but it is easy to build and runs efficiently on distributed systems. It is elastic and fault tolerant. Mesos orchestrates all components and manages computational resources [14]. Mesos is based on the principles of the Linux kernel and is the basis of three environments (Aurora Apache, Chronos and Marathon) [15]. This tool allows you to manage and organize an infinite number of machines quickly and reliably.

Akka: a library of the SMACK architecture based on the actor model. It represents a tool for developing distributed, fault-tolerant and message-driven applications. Akka uses the Java Virtual Machine (JVM) as its runtime environment, which allows development in Java and Scala programming languages. This serves as the basis for Akka in the actor model, which divides a program into simultaneous actors who are exclusively exchanging information with each other through messages [16].

Cassandra: a non-relational and distributed database manager. This handler is part of NoSQL databases [17] [18] [19]. It is scalable and fault tolerant for large amounts of data. Unlike relational databases, the database is column-oriented, which is particularly advantageous for applications that work primarily with column-based queries such as aggregations of individual columns. In the SMACK architecture, Cassandra is used to store operational data and can be used as a data source for the presentation layer [20].

Kafka: a stream processing platform it represents the data ingestion point in the SMACK architecture [21]. It is responsible for publishing and subscribing to messages. Kafka takes data from applications and streams to process it inside the stack. Kafka inspects the incoming data volume to partition it and distribute it across nodes. It is packed with several features like Automatic Fault Tolerance, high performance in distributed messages, partitioning and distribution between cluster nodes. It is also independent on the data pipeline, supports a large number of users, and processes large amounts of data [22].

The SMACK architecture essentially aims to standardize the separate batch and real-time layers in the Lambda architecture. This architecture is more of an alternative to the lambda architecture whose role was to become a standard for real-time or near-real-time Big Data applications.

3. CQRS and Event Sourcing

Events Sourcing and CQRS are methodological approaches that have been the subject of several works in the field of Software Engineering in general, and in the development of micro-services in particular. Event sourcing is a methodology in which each action generates events which are stored in an appropriate database called an event database. As for the data, they are separately in a dedicated database. In this approach, all actions on the data are not done directly via the user interface, but rather by analyzing the event matches by an event engine.

Called event header whose function is to update the database according to the generated event [23]. Command and Query Responsibility Segregation, abbreviated as CQRS, is a methodology that separates write and read operations. In this approach, the databases in which the data is written are dissociated from the databases on which the read requests are made. For a writing database, multiple read databases are generated based on business rules. And it is the latter that are used for read operations [24] [25].

In practice, event sourcing and CQRS are complementary and can be used to develop more robust micro-services capable of processing large volumes of data. It is in this context that FANSHA *et al.* implemented a microservices architecture based on the CQRS model, Events sourcing on OpenAPI, a pilot API and a pilot Event. In order to evaluate the performance of the proposed architecture, they carried out some tests according to the response time, the error rate and the throughput. Indeed, this test proved that micro-services with CQRS and Event Sourcing models have much faster performance than those of the pilot API, *i.e.* 3.7%. Moreover, it appears that the communication between the services has no effect on the error rate and the throughput [26]. KLJUN *et al.* in the quest of implementing a micro-service, made an in-depth analysis of the architecture of micro-services and the CQRS model. Thus, this inspection allowed the implementation of a micro-service based on the extension of the KumuluzEE framework. This extension allowed the integration of the Axon framework in the CQRS and Event Sourcing (ES) model [27]. Akre *et al.* have developed a CQRS and ES system that supports both event sourcing and order sourcing. Thus, they implemented multiple mechanisms of logarithmic reduction (pruning) and persistence. This made it possible to test and measure the performance of various CQRS+ES configurations. By doing so, they better understood the design principles and performance of CQRS + ES systems [28]. As for Vlček, Lukas explores the area of Enterprise Application Integration (EAI) models in combination with the CQRS architectural model. This is to specify the prerequisites for using a combination of CQRS and the design pattern of mediation or federated integration of EAI. The result of this work provides prerequisites in the appropriate use of model combinations by a given company when deciding on the final form of the EAI [29].

4. PAYI Architecture

The essential components of the Payi architecture in **Figure 3** are:

- Commands: these are basically the actions of inserting, modifying, deleting and reading data or the results of data processing;
- Bus: these are messaging brokers allowing the transport and distribution of data and events;
- Handlers, Engines and Generators: these are the data and event processing engines;
- Stores: these are the technologies for storing data and events;
- Platforms: these are technologies specific to the exploitation and analysis of data.

The architecture has the advantage of taking into account all activities related to data; from the development of data storage applications to the development of learning machine. It can therefore be used for software engineering as well as for data engineering, data science and machine learning engineering activities.

One of the advantages of the Payi architecture lies in the fact that it allows the development of batch solutions and real-time or near-real-time solutions with the same technological bricks. Indeed, the architecture unifies the development of batch and real-time solutions both at the software engineering level and at the data engineering level.

Another advantage of the Payi architecture is that it allows the processing of extreme voluminous data and the development of applications using Big Data. Indeed, event and data storage solutions can be distributed file systems (HDFS, etc.), object storage systems (S3, MinIO, etc.) or NoSQL technologies (Elasticsearche, MongoDB, etc.). Also, data and message buses can be distributed messaging brokers (Kafka, etc.) and event processing engines can be distributed computing engines (Spark, Flink, etc.).

The last advantage of the Payi architecture consists in simplifying Software Engineering thanks to Data Engineering. Indeed, all the backend consisting in

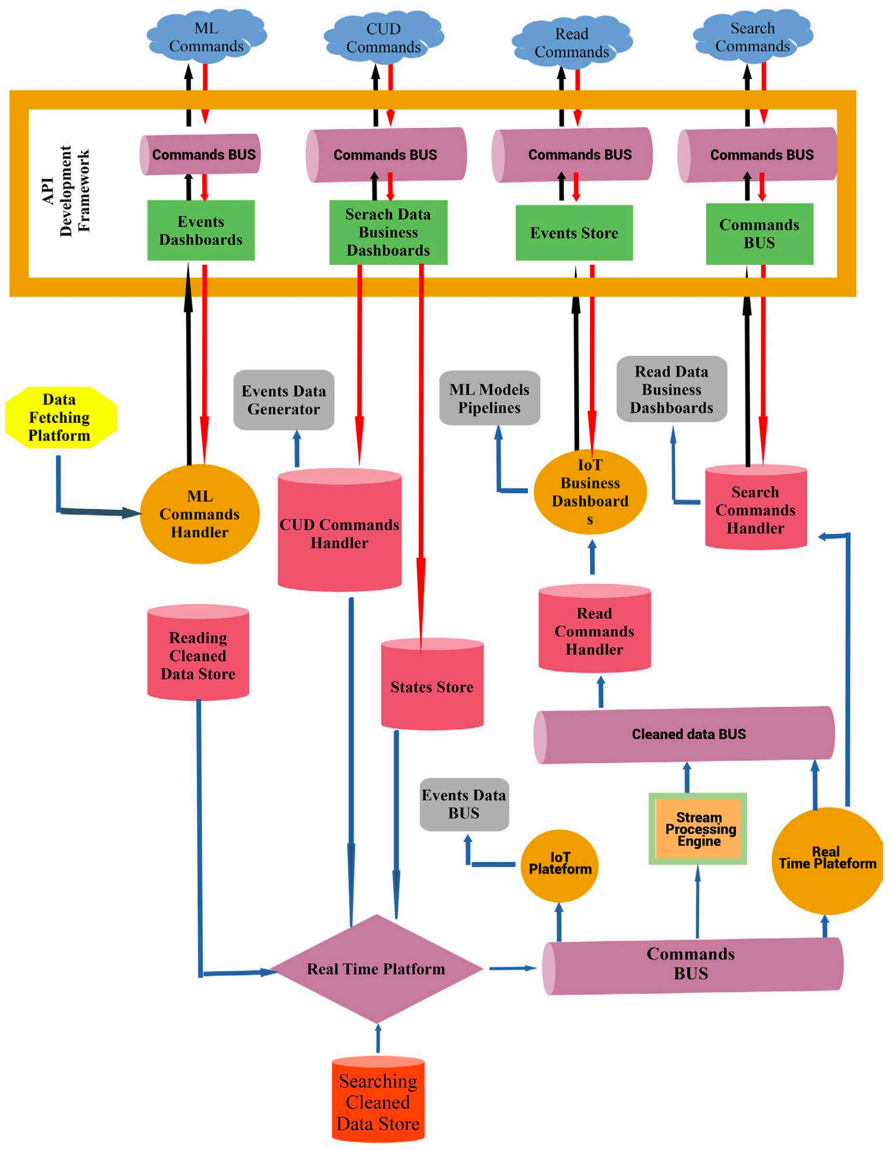

Figure 3. CQRS and events sourcing based fast data architecture.

the processing of events, they can be automated by Data Engineering technologies. Therefore, the work of Software Engineering is essentially limited to the development of the frontend of applications

5. Conclusion

We have proposed an architecture that unifies software engineering and data engineering. In reality, it is an architecture that integrates management applications with decision-making applications. Its strength is that it can be used both for development for classic applications and for real-time applications such as those of the IoT.

This architecture has the particularity of being suitable for the development of big data and fast data applications in a software engineering context. Its flexibility makes it possible to use several different big data technologies, unlike the

SMACK architecture which focuses on scala technology (Spark, Mesos, Akka, Cassandra Kafka).

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Sawadogo, P. and Darmont, J. (2021) On Data Lake Architectures and Metadata management. *Journal of Intelligent Information Systems*, **56**, 97-120. https://doi.org/10.1007/s10844-020-00608-7

[2] Sindjoung, M.L.F., Bomgni, A.B., Fute, E.T. and Chendjou, J. (2018) An Improved version of Lambda Architecture. *CARI* 2018, Stellenbosch, 14-16 October 2018.

[3] Hachad, T., Sadiq, A. and Ghanimi, F. (2020) A New Big Data Architecture for Real-Time Student Attention Detection and Analysis. *International Journal of Advanced Computer Science and Applications*, **11**, 2421-247. https://doi.org/10.14569/IJACSA.2020.0110831

[4] Big Data Architecture—Detailed Explanation. https://www.interviewbit.com/blog/big-data-architecture/

[5] Valentin, O., Jouanot, F., d'Orazio, L., *et al.* (2006) Gedeon, a Data Grid Middleware. *Data Grid. RenPar*'17: 17*th Francophone Meetings of Parallelism*. Canet en Roussillon, Oct. 2006.

[6] Saputra, F.A., Salman, M., Hasim, J.A.N., Nadhori, I.U. and Ramli, K. (2022) The Next-Generation NIDS Platform: Cloud-Based Snort NIDS Using Containers and Big Data. *Big Data and Cognitive Computing*, **6**, Article No. 19. https://doi.org/10.3390/bdcc6010019

[7] Khujamatov, H., Ahmad, K., Usmanova, N., *et al.* (2022) Fog Computing Capabilities for Big Data Provisioning: Visualization Scenario. *Sustainability*, **14**, Article No. 8070. https://doi.org/10.3390/su14138070

[8] Gu, W.Y. (2020) Improving the Performance of Stream Processing Pipeline for Vehicle Data. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science (EECS), Department of Computer Science SE-100 44, Stockholm.

[9] Rosandic, J. (2022) Real-Time Streaming Data Management, Processing, Analysis and Visualisation. University of Zagreb, Faculty of Organization and Informatics, Varaždin. https://repozitorij.foi.unizg.hr/view

[10] Batyuk, A. and Voityshyn, V. (2018) Software Architecture Design of the Information Technology for Real-Time Business Process Monitoring. *Econtechmod: An International Quarterly Journal on Economics of Technology and Modeling Processes*, **7**, 13-22.

[11] SMACK Technologies. https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781786467201/1/ch01lvl1sec9/smack-technologies

[12] Zoun, R. (2020) Analytic Cloud Platform for Near Real-Time Mass Spectrometry Processing on the Fast Data Architecture. University of Magdeburg, Magdeburg. http://dx.doi.org/10.25673/34165

[13] Atanasov, A. (2019) Cassiopeia-Andromeda-Based Protein Identification on Fast Data Architecture. University of Magdeburg, Magdeburg.

[14] Omar, A. (2021) Fast Data Application Design Architectures. Dept. Software Engineering (of Aff.), Birzeit University, Ramallah. http://doi.org/10.13140/RG.2.2.11579.54563

[15] PLASMATIC (2017) Técnicas y tecnologías Big Data para el Aprendizaje Automatico no supervisado. Entregable E3.1. Technological Institute of Informatics, Camino de Vera, Valencia.

[16] Latreider, H. (2019) Konzeption und Entwicklung einer Plattform zur Echtzeitanalyse temporaler Graphen. Hochschule für angewandte Wissenschaften Hamburg, Hamburg.

[17] Boubiche, S. (2022) Big Data Support in the Data Aggregation Process in Heterogeneous RCSF. University of Batna 2, Batna.

[18] Ryma, B. and Sonia, K. (2019) Design and Realization of a NoSQL Database under Hadoop as Part of a Smart City (Context: Flood). The Mouloud Mammeri University of Tizi Ouzou, Tizi Ouzou.

[19] Tossou, O.N.N.F. (2021) Fact Checking by Data Partitioning. African Institute for Mathematical Sciences (AIMS), Sénégal.

[20] Sánchez Piccardi,, M.L. and Palomo, L.E. (2021) Del big data al fast data: Enfoques modernos de streaming de datos para el procesamiento de datos masivos en tiempo real. *Difusiones*, **21** 38-58.

[21] Klingler, M. (2022) Confidentiality and Traceability in Publish/Subscribe Systems for Health Applications. Cryptography and Security, University of Limoges, France.

[22] Oumarou, M. (2022) Mahamadou Nouridine. Calculation of the Exact Position of a Device from Raw Data. University of Quebec at Chicoutimi, Chicoutimi.

[23] Ramón Jiménez, H. (2016) Platform for Massive Multiplayer Programming Games. Universitat Politècnica de Catalunya, Barcelona.

[24] WIPO (2018) Committee on WIPO Standards (CWS). Sixth Session. New WIPO Standard on Web Application Programming Interfaces, Geneva. https://www.wipo.int/edocs/mdocs/classifications/fr/cws_6/cws_6_6_corr.pdf

[25] Lovisetto, G. (219) A Foundation for Extensible and Decentralized Social Networks. Master's Thesis, Catholic University of Louvain, Riviere, Etienne. https://dial.uclouvain.be/memoire/ucl/object/thesis:19458

[26] Al Fansha, D., Setyawan, M.Y.H. and Fauzan, M.N. (2021) Load Test pada Microservice yang menerapkan CQRS dan Event Sourcing. *Journal Buana Informatika*, **12**, 126-134. https://doi.org/10.24002/jbi.v12i2.4749

[27] Kljun, M. (2020) Arhitekturni Model Implementacije Vzorca CQRS v Okolju Mikrostoritev. https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=122409

[28] Akre, M.B. (2020) Event Log Pruning in CQRS Systems. Institutt for datateknologi og informatikk, Trondheim.

[29] Vlček, L. (2018) CQRS and EAI Integration Design Patterns—Assumptions of Mutual Combination. Prague University of Economics, Prague.