Tampere University of Applied Sciences

Near Real-time Clickstream Analysis

A Journey in Big Data Systems and Architectures

Salvatore Corrao

MASTER'S THESIS
April 2019

Master's Degree in Information Technology

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master's Degree in Information Technology

SALVATORE CORRAO:
Near Real-Time Clickstream Analysis
A Journey in Big Data Systems and Architectures

Master's thesis 59s pages, appendices 0 pages
April 2019

The clickstream analysis focuses on the records generated while a user clicks on a web page. This field is nowadays part of the Big Data phenomenon and uses near real-time software implementations.

The aim of this thesis was the implementation of a near real-time Big Data infrastructure that can uphold a clickstream analysis. This work limited the clickstream analysis implementation to mainly the user sessionization function. The infrastructure architecture design used open-source software to enable five core data capabilities which are ingestion (consuming the click records), transformation (data cleaning, user sessionization, user agent enrichment), storage, analytics (insights) and visualization (for presenting accessible insights).

The implementation was run interactively, moving step by step through different technical options. The iterations followed a simple scheme. What is easy to install, to configure, and to test? What is general enough to solve more complex requirements? What can be removed? In particular, the sessionization algorithm implementation easiness was a benchmark to compare the various infrastructure iterations.

There were four design iterations and four infrastructure implementations. A first realization was a zero-coding infrastructure. A second phase delivered a more capable parallel data processing component based on Apache Spark, a central framework in this work. The next implementation simplified the data storage and started the exploration of the Apache Spark streaming features. The last experiment showed the possibility to process streams of clickstream data, coming continuously from a weblog, with low latency by using Apache Spark Structured Streaming.

Spark Structured Streaming has a few SQL limitations that require adapting the algorithms and the processing sequence. However, Spark Structured Streaming is in its infancy, and there are good reasons to believe that it is going towards fewer limits. Besides the central place of Spark, many technologies can set new directions or bring improvements to this thesis; for example, specialized databases as the clickstream records are a time-series and also a graph.

---

keywords: clickstream analysis, streaming, sessionization, apache spark, spark structured streaming

CONTENTS

ABBREVIATIONS AND TERMS

API	Application Programming Interface
BI	Business Intelligence
Apache Cassandra	A NoSQL database
CPU	Central Unit Processing
DSL	Domain Specific Language
ELK	Elasticsearch, Logstash, and Kibana
ELT	Extract Load and Transform
ETL	Extract Transform and Load
FTP	File Transfer Protocol
Apache Hadoop	A collection of open-source Big Data software
HDFS	Hadoop Distributed File System
Apache Hive	A data warehouse system built on Apache Hadoop
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Services (Microsoft)
I/O	Internet Information Services
JSON	JavaScript Object Notation
NSA	National Security Agency
Apache Pig	A platform with a high-level language for creating Apache Hadoop programs
Proxy	A computer system in-between the client and the servicing server.
RDBMS	Relational Database Management System
REST API	An API that uses HTTP requests to get access to data.
Apache Spark	An open-source embarrassingly parallel computing framework.
TAMK	Tampere University of Applied Sciences
UTC	Coordinated Universal Time
WC3	World Wide Web Consortium
WCA	Web Content Accessibility
Apache YARN	Hadoop Cluster resource manager

1 INTRODUCTION

The present master thesis deals with the design and development of a modern, scalable solution to analyze website users' clicks from the web server logs. The focus is to deliver the information in near real-time with the tools of the so-called Big Data. One of these tools, Apache Spark, is central to this thesis.

1.1 General Context

Since the beginning of the World-Wide Web, many organizations have wanted to quantify or monetize the insights they can get on the audience of their websites. These insights can help to design a better website, measure the effectiveness of a marketing campaign, and "*predict what products the user is likely to purchase*" (Wikipedia, Click path) using previous purchasing data. Information is valuable; for example, a successful second-hand car website can also make a substantial profit by selling quantitative insights on people's favorite cars by model, by color or by demographic dimensions (such as gender, age, region). Nowadays, algorithms are continuously running on the websites back-ends, for example, to guess someone mood or gender. Those web monitoring tools allow the business to gain an understanding of the nature (e.g., human, bots), the profiles (e.g., customer, prospect, competitor), the behaviors of the visitors of their websites.

The web monitoring tool tracks which page the user request, how much data the web server serves, what is the previous page visited. The clickstream (or click path) is the path the visitor takes through a website. Clickstream analysis or clickstream analytics is collecting, analyzing and reporting information about the click path (Wikipedia, Click path).

1.2 Problem Description

The volume of clickstream data gathered is often substantial. A publicly available weblog for the 1998 World Cup Web site gives 1,352,804,107 requests between April 30, 1998, and July 26, 1998. Today, YouTube gets 15 billion visits by month (Youtube, Press) with an average of 8 pages by visit (192 billion requests). As

data were getting bigger and bigger, processing became troublesome for conventional systems (such as a relational database) and algorithms because of the high computation cost. Introducing distributed data processing platforms such as Apache Hadoop helped with the processing of a considerable amount of data with low fees and in a reasonable amount of time. (Google inspired the Big Data practice by publishing a paper on the Google File System in 2003).

Today, businesses are trying to uncover opportunities to capture and respond to business events faster and more systematically than ever. The requirement is to optimize the duration between an event and its consequential action to get a greater benefice or value. The right-time would be that the business case specifies an absolute required freshness for the data. For example, security requires an action within seconds when some trend information is weekly. However, the daily scheduling (traditionally used in data warehousing) can no longer be the only option to offer as today's business users emphasize a lower latency between the events and the decision makings (human or automated decision).

This thesis presents a journey (including several software design iterations) into highly scalable systems for storing and analyzing clickstream data as the stream of clicks generates log entries on the web server.

1.3 Near Real-time

“A computerized real-time system is required to complete its work on a timely basis. Typical applications are digital control, command and control, signal processing and communication systems” “The main characteristic of real-time systems is the behavioral predictability. Timing constraints will be met whatever happens in the system.” (J. Goossens, P. Richard, 2004)

One can define real-time in a very mathematical way, but its everyday usage is unclear, and its meaning varies with the context. For many persons, it means super-fast; for others, it denotes the ability to processing the data as the system is receiving it.

The wording near real-time or even recently right-time introduces even more imprecision. However, a commonly accepted use when the delay involved by the processing of the data is in the second range to a few minutes. (Wikipedia, Real-time computing)

This thesis uses the near real-time with the meaning of giving a result by avoiding unnecessary delays, providing the result in seconds to minutes with no guaranty of the execution time in case of a no favorable scenario (e.g., with a burst of visits on the website).

1.4 Motivations and Contributions

The big data journey is a difficult task because of various challenges.

- There are several large ecosystems, each of them with many technologies: processing frameworks, storage systems, data flow languages, tools for ingesting the information, and non-relational databases.
- The subject is trendy, making it hard to distinguish the core platforms.
- There is no absolute consensus on the best solution.
- Many solutions depend on the skills and affinity of persons.
- Behind even the open source systems, there are significant and antagonistic financial interests.

So big data is difficult at first sight, and it makes it an exciting challenge.

1.5 Clickstream Analysis and Privacy

While there are many benefits that the mining of the weblogs brings, a definite drawback is a potential for severe violations of privacy. The dataset used for the practical part of this thesis was missing the IP address of the browser, and no cookie information was available. Although less accurate regarding the definition of the user, there is still enough information to create a user footprint but not enough to identify an individual.

1.6 Outline

This thesis contains the following chapters:

- Chapter 2 presents some facts on clickstream and especially the sessionization, which is a central concept in clickstream analysis.
- Chapter 3 lists some considerations on Big Data architectures that led the implementation.
- Chapter 4 is about Apache Spark, a massively distributed application framework and engine, the core of the realized solution.
- Chapter 5 shows the iterative and suggestive process to find architecture.
- Chapter 6 draws a few conclusions.

2 CLICKSTREAM ANALYSIS

2.1 Introduction

A Clickstream makes up the visible part of the user online behavior. On the web servers, there is a history of what human or bot click while browsing the Internet. Clicking anywhere on a web page saves information (such as the Internet browser used, the page requested) on the user client and the web server. Nowadays, the most popular form of click tracking uses a JavaScripts tracking code, which is inside the web-pages. Using a client-side method is more accurate. The reason is that the web server can miss client requests when there is a proxy or a cache between the client and the web server.

Modern methods of Clickstream Analysis can answer many questions, for example:

- What is the user gender or his/her emotional state? (Machine Learning Algorithms)
- Is the user intention to buy something?
- How do visitors travel inside of the site?
- Which pages are loaded only sporadically?
- What is the visit-to-purchase ratio?

This thesis narrows the data source to only web server log-files. This approach to click-stream analysis was historically the first (before Google and others JavaScripts), and it has the practical advantage to focus on the data processing rather than on the complex data-collection and exchange of the modern scrips and cookies that track us on the internet. Also, the discussion in this chapter focuses on the analysis of one crucial user dimension: the session. The session tells about the basic user behavior: how long the user stays on the website before leaving, how many times he/she comes back.

The experimental data comes from seven days weblogs from a Finnish multinational website, and it is worth underlining that no IPs are identifiable. (The web server is behind a proxy).

2.2 Web Server Log Format

A web server records every access to a web page in its logs. A weblog record is a structure that follows a pre-defined format. The choice of the information in the log and the binary/text format of the log file depends on the web server brand and its configuration.

Below in Table 1, an anonymized record from the Microsoft IIS server logs.

2017-09-09 01:04:59 GET /folder/page.htm - 10.1.17.240 HTTP/1.1 MobileSafari/602.1+CFNetwork/811.5.4+Darwin/16.7.0 - 200 0 0 19161 426 171

Table 1. IIS web logline decoded

Field	Record string	Description
Date	2017-09-09	The date on which the server received the request.
Time	01:04:59	The time, in coordinated universal time (UTC).
Method	GET	The HTTP method such as GET for requesting a web page.
URI Stem	/folder/page.htm	The page/object requested by an HTTP method.
URI Query	-	The query such as for forms or REST API.
Client IP Address	10.1.17.240	The user IP address. (Here it is a local network.)
Protocol Version	HTTP/1.1	Protocol version for HTTP.
User Agent	MobileSafari/602.1+CFNetwork/811.5.4+Darwin/16.7.0	The browser type that the client used.
Referrer	-	The site from which the user is coming.
HTTP Status	200	The HTTP status code is the code returned to the client, like the famous 404 when a page is not found.
Win32 Status	0	The Windows status code.
Substatus	0	A second Windows status code.
Bytes Sent	19161	How many bytes the server sends to the user client.
Bytes Received	426	The number of bytes received by the server.
Time Taken	171	The time to complete the user request in milliseconds.

The minus sign (-) is for an empty field.

2.3 Weblog Information

The information in the web access logs allows us to construct or identify specific levels of data abstraction: user, server session, episode, clickstream, and

pageview. The W3C Web Characterization Activity published the definitions of the terms relevant for analyzing navigation behavior. (W3C, Terminology)

A **user** is an individual who is accessing files on web servers through a web browser. A user can access the Web with different computers.

A **pageview** is all the files that make the display of a page on the user's browser.

A **clickstream** is a temporal sequence (a series) of pageview, sometimes incomplete because the user request is served by his/her browser cache or proxy-level cache in the network provider.

A **user session** is a clickstream for a single user across the entire Web. Abstractly, a user is visiting several websites on the same time window, and practically, no private company can access all the weblogs on all websites.

A **server session** is a set of pageviews in a user session for a particular site, commonly called a visit.

2.4 Sessionization

A session/visit comprises the time-series of the user's clicks from the moment that the user enters the website and clicks on the site pages, to the moment he/she exits. Often the users do not register on an Internet website. Therefore, the applications make an approximation by using some level of user fingerprinting (Panopticlick, Browser Fingerprinting). An approach for discriminating amid unique visitors is using client-side cookies, but users sometimes disable cookies to keep their privacy or to avoid malicious code execution. IP addresses are not sufficient because of the network providers that assign rotating IP addresses to their customers. Therefore, a user is frequently a function of several pieces of information available in the server log: IP address, user agent, referrer (the URL the user comes from). Sometimes, the user is the registered user name or a session cookie if recorded.

The sessionization is a transformation that segments the pageviews of each user into sessions. Each session is a single visit to the website. A practical method for sessionization is to assume that the time spent on a single page must not exceed a threshold. (Osmar R. Zaiane, Jaideep Srivastava, Myra Spiliopoulou, Brij Masand, 2002)

In many commercial products such as Google Web Analytic, two clicks of the same user separated by 30 minutes make up a different session for the user. (Google, How a web session is defined in Analytics)

3 A JOURNEY IN BIG DATA ARTEFACTS

3.1 Introduction

There is no surprise in the proliferation of solutions and businesses trying to address different aspects of Big Data. The hardware and the technology to store and access large sets of data on commodity hardware are affordable and accessible. Starting a Big Data journey is about being overwhelmed by fancy product names, new technologies, and terminologies, to cite a few: Hadoop, Hive, Pig, Spark, Cassandra, Lambda Architecture, immutability. Figure 1 shows a few of those Big Data solutions.

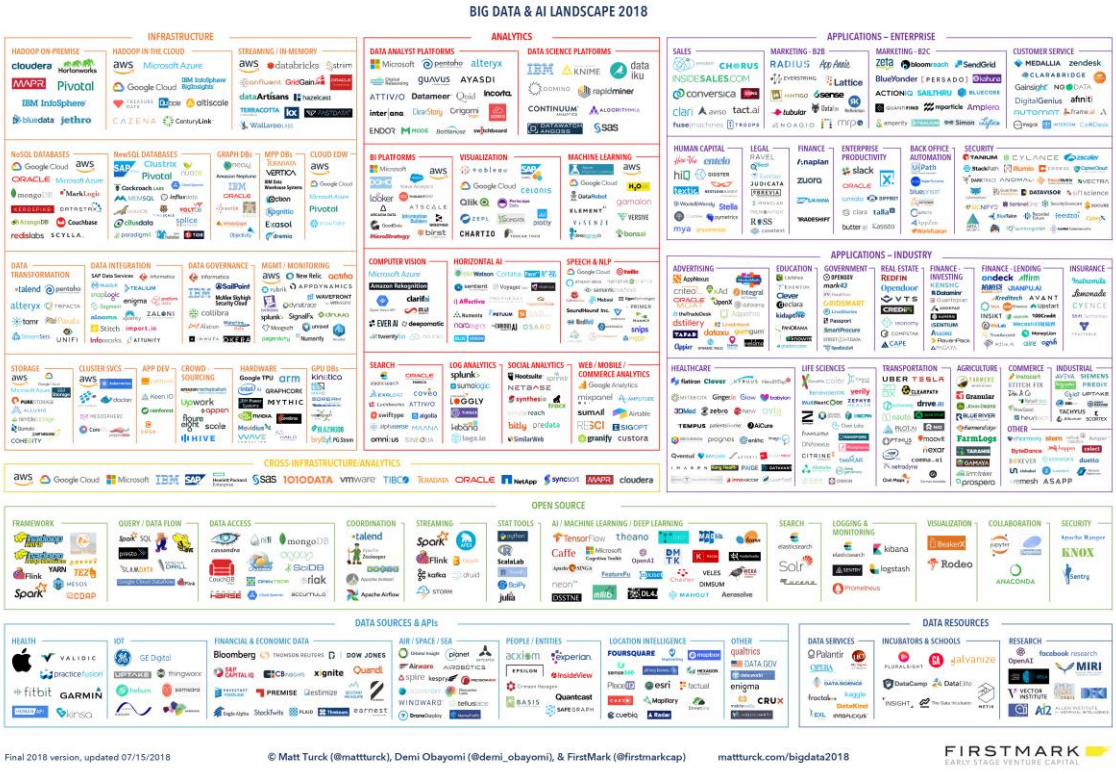

Figure 1. Big Data Landscape 2018 (http://mattturck.com/wp-content/uploads/2018/07/Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png)

Big Data is not a single technology but a cluster of many techniques and tools that the development of various business scenarios use. There is inherent uncertainty as the field is continuously expanding to new concepts and technologies.

This thesis Big Data challenge, using clickstream analytics as a proof of concepts, is to find out which principles, technologies, frameworks, languages work bests and allow fast learning, quick implementation and efficient iterations towards base level knowledge in a growing and complex domain.

3.2 Considerations on Big Data Systems

Big Data starts where traditional systems, based on relational databases, break under the combined velocity, volume, and variety of the internet information area.

Introductions to Big Data often cite the following V's as essential attributes of the big data:

- Volume, the quantity of data produced and stored. The current amount of data is staggering as an example every day people watch over a billion hours of video on YouTube. (Youtube, Press)
- Velocity, the pace at which data is generated, produced, created, or refreshed.
- Variety, the structure of the data. Structured data but also semi-structured and mostly unstructured data (e.g., clickstream logs, sensor data, audio, image, video, social media, log files).

Those 3 V's are the core of what define big data. Authors and experts have added a few more V's over the years. Best-selling author Bernard Marr says that only five V's matter, adding Veracity as the reliability of the data and Value as the ability to turn data into value (Bernard Marr, March 2015).

Traditional systems handle these constraints by using different techniques such as queuing transactions (reducing the pressure on the back-end) and grouping insert/update requests (avoiding sending several single requests), sharding the

database (splitting tables across several databases), horizontal partitioning (re-distribution of the load across different systems/application servers and databases).

Those techniques are not outdated and are even still in use in Big Data. However, in a traditional application, the database is not aware of its distributed nature, and scaling the solutions hits several issues:

- The developer code has to manage the underlying complexity of the database shards, moreover, the logic that distributed the data across several shards.
- As the number of machines is going up; the event of a disk failure is increasing,
- Network bandwidth issues when processing large sets of data.
- Mistakes are difficult to correct (it's always challenging to restore a backup if the data get corrupted).

The traditional systems push the complexity of handling large datasets to the persons operating the database or developing the software.

The big data technologies are a change:

- The technology handles shards and data replication. Freeing the developer who can focus on the application logic.
- Scaling is as simple as adding new machine nodes.
- The different frameworks optimize data throughput and minimize network transfer.
- The raw data are immutable, which in principle allows rebuilding everything in case of a mistake.

Eric Brewer's CAP theorem is a fundamental limit of any networked shared-data system. A distributed system must do trade-offs.
"The CAP theorem states that any networked shared-data system can have at most two of three desirable properties: consistency (C) equivalent to having a single up-to-date copy of the data; high availability (A) of that data (for updates); and tolerance to network partitions (P)" (Eric Brewer, February 2012)

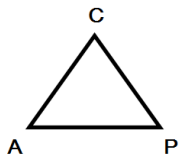

Figure 2. CAP Theorem trade-offs

"The "2 of 3" formulation was always misleading because it tended to oversimplify the tensions among *properties (Eric Brewer, February 2012)"*. A representation such as Figure 2, can misinform on the pivotal role of P. More precisely, the theorem states that when a network partition (P) arises and for example, cuts off the network link between 2 nodes, the system can be consistent (C) or available (A) but not both.

Example of arbitration implied by the CAP theorem: two users who do the same search on the same search engine can get different results. The search engine business considered that this disadvantage is less severe than not having any results at all.

3.3 High-Level Architectures

Presentations of Big Data are often from the angles of environments like Spark or Hadoop, or platforms such as Hortonworks or Cloudera. There are always considerations specific to each of these technologies, but the question that arises, is if there are any reusable practices in Big Data, regardless of the particular (commercial or not) environments?

A classic architecture comprises an incrementally updated central database. Even without large datasets, this architecture is often problematic because:

- At a certain point, scalability is challenging to get.
- There is a risk of data loss because of unavailability or human errors.
- The complexity of maintenance and use increases with time.
- There is significant latency before getting the data ready to be consumed with simplicity and adequate performance.
- It is hard to get the same scalability, robustness, and simplicity in one place.

Because the centralized approach to storage and processing is no longer enough, it is then necessary to use a distributed method on clusters of machines. There is a functional need to separate storage, consumption, and complex processing. Several pioneers designed Big Data architectures, such as the Lambda architecture, to solve complex problems requiring the intervention of several technologies. These architectures are something similar to design patterns in object languages that ensure the reusability and the sustainability of the code.

The following Big Data architectures address the processing, storage and data analysis:

- Lambda architecture
- Kappa Architecture
- Data Lake Architecture

3.3.1 Lambda architecture

The programmer Nathan Marz wrote a popular blog post (Nathan Marz, How to beat the CAP theorem) and book (Nathan Marz, James Warren, 2015) describing what he called the Lambda Architecture. Not all information is equal, as most of the time, a program derives data from another piece of information. At the root of the data lineage, the master dataset is the rawest data, and it gets the benefice of being immutable. In case of error leading to data corruption, all views are rebuilt from the master dataset. The Lambda architecture performs batch processing and real-time processing simultaneously, making it possible to merge batch processing with streaming input (real-time) processing.

The Lambda architecture (see Figure 3) transmits entering data (1) in two different routes simultaneously: a batch-oriented path (2) and a real-time circuit (4). A query result (5) is the merging of the indexed data (3) coming from the batch layer and available until time T minus some delay δ and the real-time data available for that period T - δ to T. A batch can process a big chunk of data with a more accurate algorithm. Its setback is the latency (slowness), which is why the speed layer provides recent data.

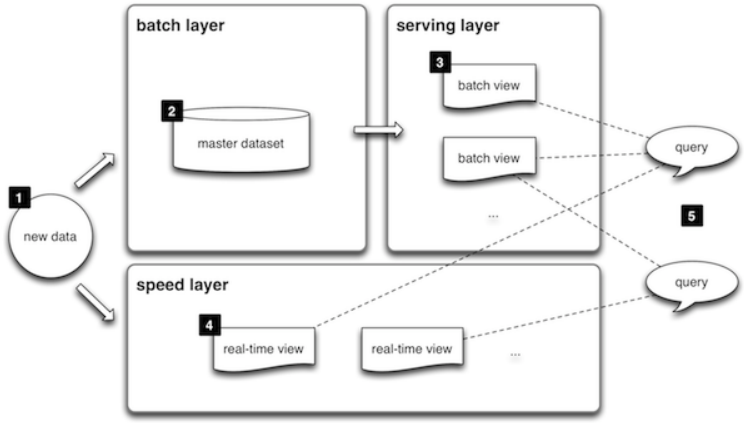

Figure 3. Lambda Architecture (http://lambda-architecture.net)

Lambda architecture aims to get a complete view of the data.

Batch layer

The batch layer stores a replicated copy of the master data set. The copy is immutable and continually growing (= store everything and capture all changes). Keeping the data in a raw format ensures that there is no data loss, as there is the possibility to redo calculations and to recreate views. Another advantage is greater flexibility because, if an algorithm improves, the reprocessing of the raw data gives results of a higher quality. This opportunity is impossible if someone preserves only the results. The batch layer deals with the calculation of the batch views. These views result from the transformation and derivations of the raw data, continuously rebuilt or updated as new data arrives.

Service layer

The service layer indexes the batch views as soon as the batch layer finishes the computation. As for the batch layer, there is the distribution on the machine clusters to ensure scalability. There must be, therefore, a balance between the amount of calculation in the batch layer and the performance of the service layer.

Performance layer

There may be latency during storage and pre-calculation of batch views before new data becomes available. The purpose of the performance layer is to make new data available as fast as possible. The performance layer is a kind of batch layer that produces real-time views updated as it receives new data. Using incremental algorithms (e.g., approximate counting algorithm) is sometimes necessary to comply with the complex latency constraints. Although, this complexity is for a few hours of data only. Once the data ages a few hours, they are available in the batch and service layers and are not more required in the performance layer. Typically, service layer queries that require low latency combine the batch and real-time views to run.

In sum, the Lambda architecture is a Big Data architecture practice that separates storage, consumption and the complexity of getting low latency. The approach does not require specific software, and with some variations in the detail of its implementation, the software engineer can use different Big Data technologies.

3.3.2 Kappa A Streaming Architecture

The Kappa architecture (Figure 4) overcomes the complexity of the Lambda architecture by merging the real-time and batch layers. It is less complicated than the Lambda architecture that requires synchronization and simultaneity of the batch and streaming capacities. Jay Kreps introduced it in his article "Questioning the Lambda Architecture" (Jay Kreps, July 2014).

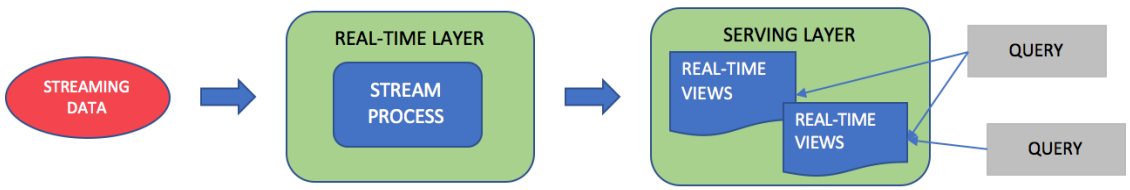

Figure 4. Kappa Architecture

Kappa architecture is when the processing to carry out is whenever batch and the streaming layers are the same. Misuse of the architecture is when streaming generates poor results because of approximation algorithms or when complex

quality tradeoffs are necessary to avoid large processing operations. In those cases, having a batch layer is a more versatile solution.

3.3.3 Data Lake Centered Architecture

The data lake is an architecture that emerged with Big Data technologies, allowing cheap storage of large volumes of data. Businesses get with it, storage systems to accommodate many types of data: structured (e.g., data with rows and columns), semi-structured (e.g., JSON documents), unstructured (e.g., videos). Unstructured is a word for something that the relational data model cannot define easily. In fact, *"There's no such thing as unstructured data"* (Chuck Densinger, Mark Gonzales, 2016). Although not the only one, Hadoop (based on the distributed file system HDFS) remains the most used reference framework of a data lake, especially when systems are on the company data center premise. Solutions storage services such as Microsoft Azure Data Lake and Amazon S3 are accessible when a company is not reluctant to a cloud solution.

In a data lake architecture (Figure 5), the data can come from multiple sources like logs, web services. The different ingestion systems are consuming the data, then inserting them into the data lake (Hadoop, Azure Data Lake, and Amazon S3). Once the data is saved, upstream processes can ingest or make queries.

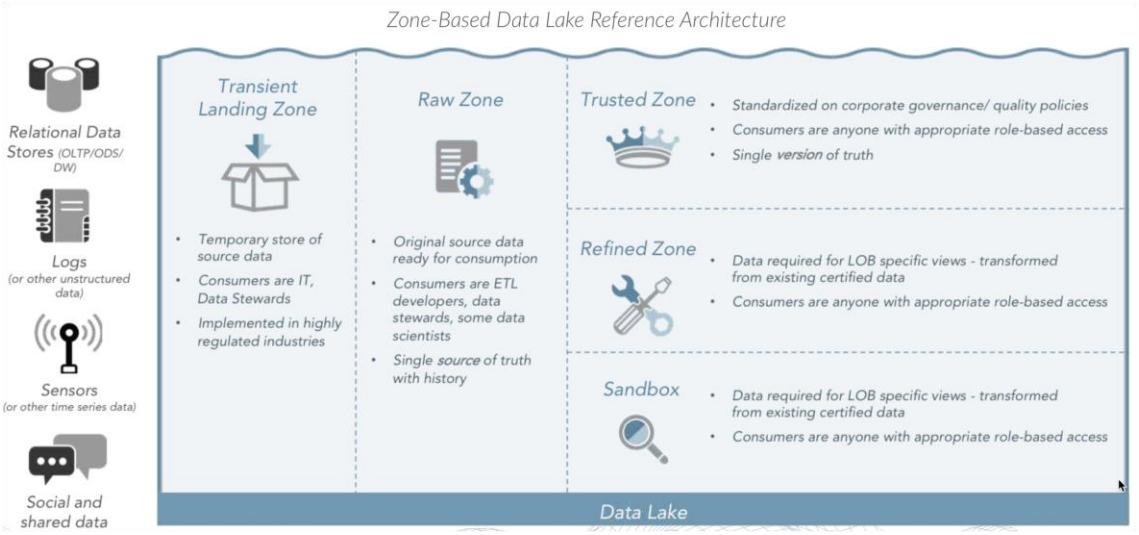

Figure 5. Zone-Based Data Lake Reference Architecture (https://resources.zaloni.com/cloud/the-data-lake-reference-architecture)

3.3.4 Conclusions

Choosing the ideal data architecture is difficult, and the answer to specific problems is sometimes to merge several architectures. Likewise, the choice is subject to the answer to questions about scalability, learning curve, reduction in production times and costs.

3.4 Business Intelligence and Big Data

Big Data does not replace but extends (see Figure *6*) the so-called data warehousing and business intelligence (BI).

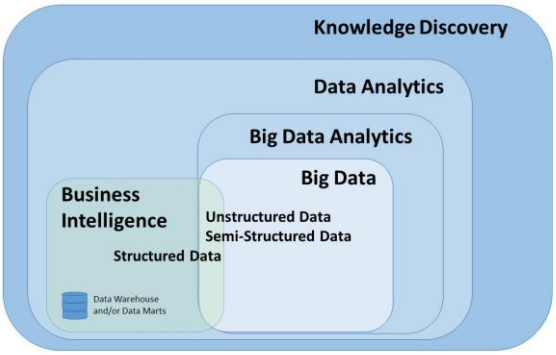

Figure 6. Business Intelligence vs Big Data

(https://www.linkedin.com/pulse/differentiating-business-intelligence-big-data-analytics-nedim-dedić)

3.4.1 Integration approach

The development of a traditional ETL (Extract, Transform, Load) data warehouse is lengthy and costly. An element of explanation is that perfect knowledge of the data is essential to store a piece of information in a database table. Conversely, the ELT (Extract, Load, Transform) approach significantly eased the trouble by dividing the processing into two parts. Data are first integrated into their raw state and kept in a structured environment. A second optional step may then transform the saved data into a consumption structure.

ELT permits an integrated and consistent approach but also allows faster storage of the raw data in the warehouse. A close concept is schema-on-read which is delaying data modeling and schema design until long after a process started the load the data and only when there is a valid business case. A data lake can over the years become bloated with never used data, as always, some governance principles are not harmful.

A benefit of the ELT approach is the possibility of adding new extractions quickly. However, it can lead to multiple extractions and replications of the same. ELT alone is neither a unified vision (according to the needs of the moment, the developer adds an extraction) nor a harmonization process.

3.4.2 BI Integration Opportunities

There are different ways to integrate big data into the existing BI environment. The Big Data is most often experienced as an extension of current BI environments to enjoy a higher speed of treatment, a more significant variety of data, or merely to take advantage of a low-cost, flexible environment as a point of entry into a data warehouse.

Parallel integration (Figure 7**)**: all conventional sources continue to feed a traditional data warehouse. While Big Data sources feed a big data environment in parallel, subsequently, the integration takes place either in a conventional consumer-oriented data warehouse or directly with the consumer tools.

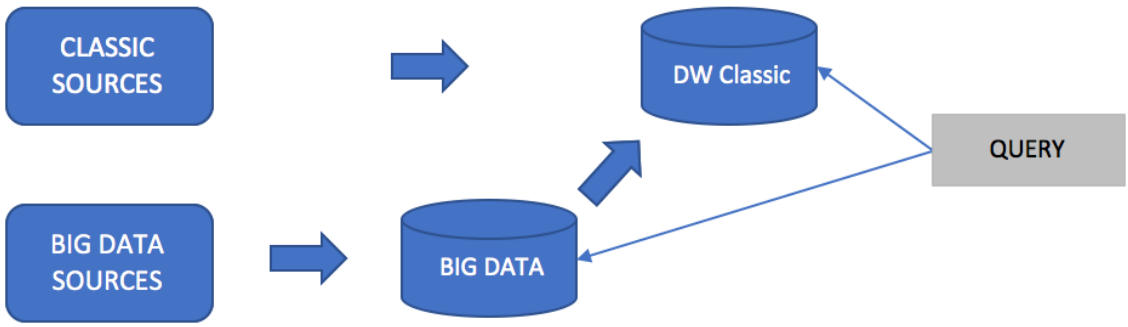

Figure 7. BI & Big Data, Parallel Integration

This approach is less intrusive and therefore easier to implement. In return, integration is less functional because there is over one way to interface with sources

and more inclusion for downstream consumption. An organization that already has a substantial investment in a traditional BI environment and wants to merge it by incorporating Big Data sources often prefer this integration.

Serial integration (Figure 8**)**: it integrates all sources into the Big Data platform, which makes the integration of sources better. However, ETLs from conventional sources are to convert to ELTs; This kind of integration requires more components but allows for consumption from a more traditional environment. It can be a transition to a possible Big Data architecture alone.

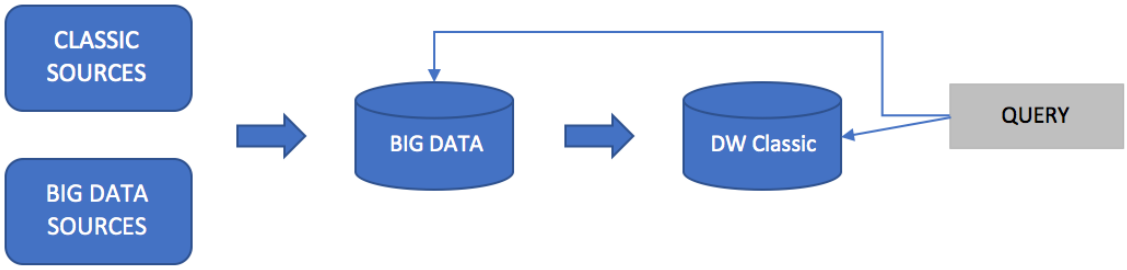

Figure 8. BI & Big Data, Serial Integration

Full BI Integration (Figure 9**)**: At the limit, it is possible to use only the big data environment as a data warehouse and consume directly from it. Most known consumer tools now have connectors that make queries to data lake more or less transparent compared to a typical data warehouse. In principle, it is the best integration possible, but it involves the additional conversion of existing consumer treatments. Besides, the maturity of the BI connectors remains a factor to consider.

Figure 9. BI integrated in Big Data

3.5 File Formats/Data Storage

In the Big Data context, there are many technical solutions for storing data. As needed, data can be stored in files on a distributed file system (DFS) or managed by a specialized tool (e.g., Hive, HBase, Cassandra). The choice between all these solutions depends on the data access profile (random access, sequential reading, append updates, row-level updates), desired performances for the critical operations and the richness of the proposed functionalities. For example, a Parquet file provides excellent performance for sequential reads but does not allow row-level updates.

The file format used has a significant impact on query performance. Some formats include compression support that decreases the space occupied on the occupied disk and consequently the I/O readings and CPU utilization required to deserialize the data. The amount of needed I/O and CPU resources can be a limiting factor in terms of the performance of running a query. The most used formats are:

- **Row Columnar File** (RCFile), Facebook initially developed it to add an SQL data warehouse system (Apache Hive) on top of their Hadoop cluster. RCFile aims to offer fast data loading, quick query processing, efficient use of storage, adaptability to dynamic workloads
- **Apache Avro** is a file container and a schema-oriented binary data serialization format. This file format can be split and compressed. The programs can save data in an Avro file with its schema and the data type descriptions.
- **Optimized Row Columnar** (ORC) combines the performance of RCFile with Avro's flexibility. Hortonworks initially developed it to overcome the perceived limitations of other available file formats.
- **Parquet** results from the combined efforts of Cloudera, Twitter, and Criteo, then donated to the Apache Software Foundation. Google Dremel document (Sergey Melnik, 2011) inspired its design. Parquet allows the developer to work with complex and nested data structures and allows efficient column-level coding.

3.5.1 Parquet file format

Parquet is a binary and column-oriented file format for large analytic queries. This file format (see Figure 10) is particularly suitable for projections (selecting fields) and aggregation operations such as SUM and AVG that must process all data for a given column.

A parquet file is, in fact, multiple data files. Each of the single files contains the values for a series of rows (row groups). Within a file, the structure organizes the values of each column to be adjacent, thus allowing a good compression and CPU cache optimization. Although Parquet is a column-oriented format, it keeps all data related to a row in the same data file, to ensure that all the columns of a row are available for processing on the same node. Queries launched on a Parquet table can retrieve and analyze column values quickly and with minimal I/O.

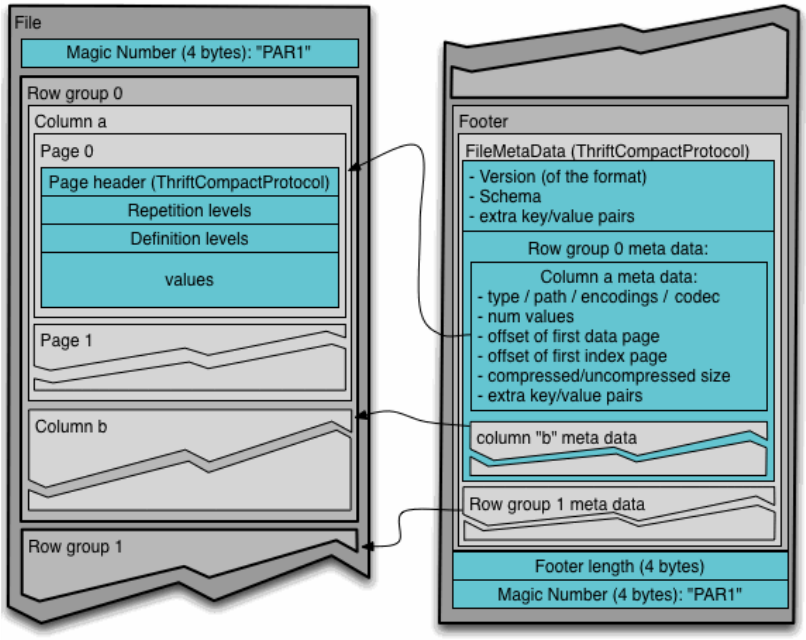

Figure 10. Parquet, File Format Detailed Structure (https://parquet.apache.org/documentation/latest/)

A column store is a file or in-memory structures optimized for the analytical queries, which is typically the selection of a few columns/dimensions along which the result aggregates some amount or quantity (e.g., average salary by department).

As an example, the following logical table (see below Figure 11) can be stored differently.

A	B	C
A1	B2	500
A2	B2	200
A3	B3	450
A4	B1	100
A5	B2	600

Figure 11. A Row Table

Traditional “database” storage of tables is a row store. They store each row of data one after the other in a perfect sequence as follows. A column-oriented structure arranges them one column at a time (Figure 12).

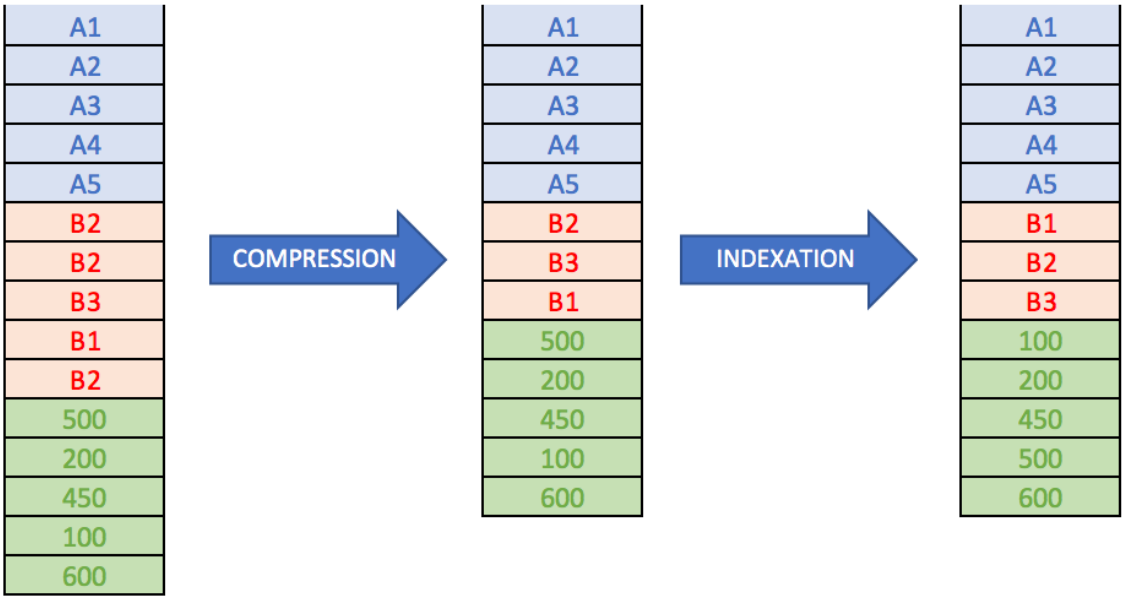

Figure 12. The Table Column Storage

There are several advantages:

- A better compression, as similar data usually have a better compression ratio.
- Better I/O because of smaller files (compression) and also because often queries select only a subset of the columns.
- Effective use the modern CPU as the data have a higher probability of being in the CPU caches, instead of branching to retrieve the data from the main memory.

3.6 Logical Architectural Pattern

The following paragraphs compose a holistic view on big data which try at first to be products and services agnostics.

Data usually goes through a life cycle, and Big Data is no exception. Challenges are not only inherent to the characteristics of data (e.g., volume, variety, velocity) and how to process it, but also at the level of the information management such as security and privacy (see Figure 13).

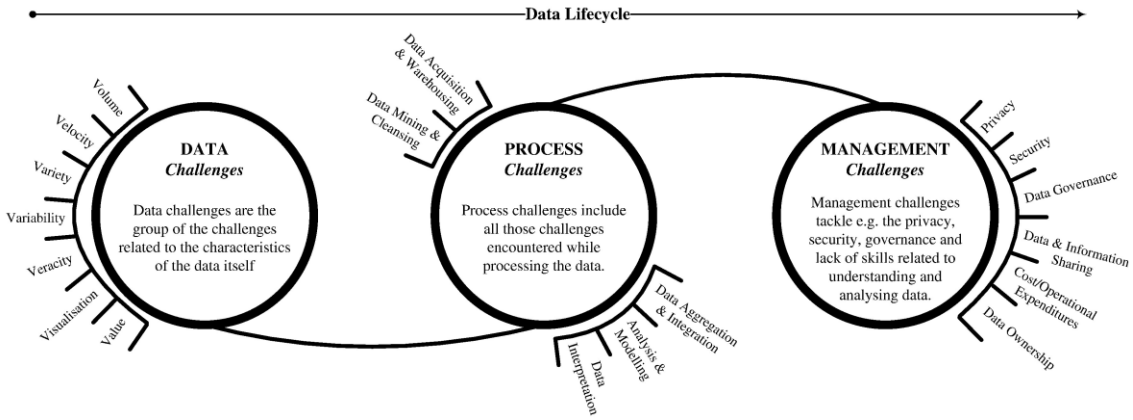

Figure 13. Data Lifecycle
(https://doi.org/10.1016/j.jbusres.2016.08.001)

A robust architecture addresses data challenges and processing capabilities. Further, it must provide:

- The components to support the data life-cycle from data creation and its organization through transformation, integration, consumption and finally archive or deletion.
- Accountability on the data flows through an organization which has to ensure sharing, privacy, and security.
- An efficient cost.

For the components of a Big Data architecture, most of the reference architectures are from the well-known names (Facebook, Uber, LinkedIn, Netflix to cite a few) and based on their product developed internally (then open-sourced and

proposed to the Apache organization). Another source is the Lambda and Kappa architectures.

The ideal reference framework would be a "technology independent reference architecture for big data systems, which is based on analysis of published implementation architectures of big data use cases. An additional contribution is classification of related implementation technologies and products/services, which is based on analysis of the published use cases and survey of related work. The reference architecture and associated classification are aimed for facilitating architecture design and selection of technologies or commercial solutions, when constructing big data systems." (Pekka Pääkkönen, Daniel Pakkala, February 2015)

The Figure 14 shows Pekka Pääkkönen (2015) ideas with a comparison with well-known data warehouse functions.

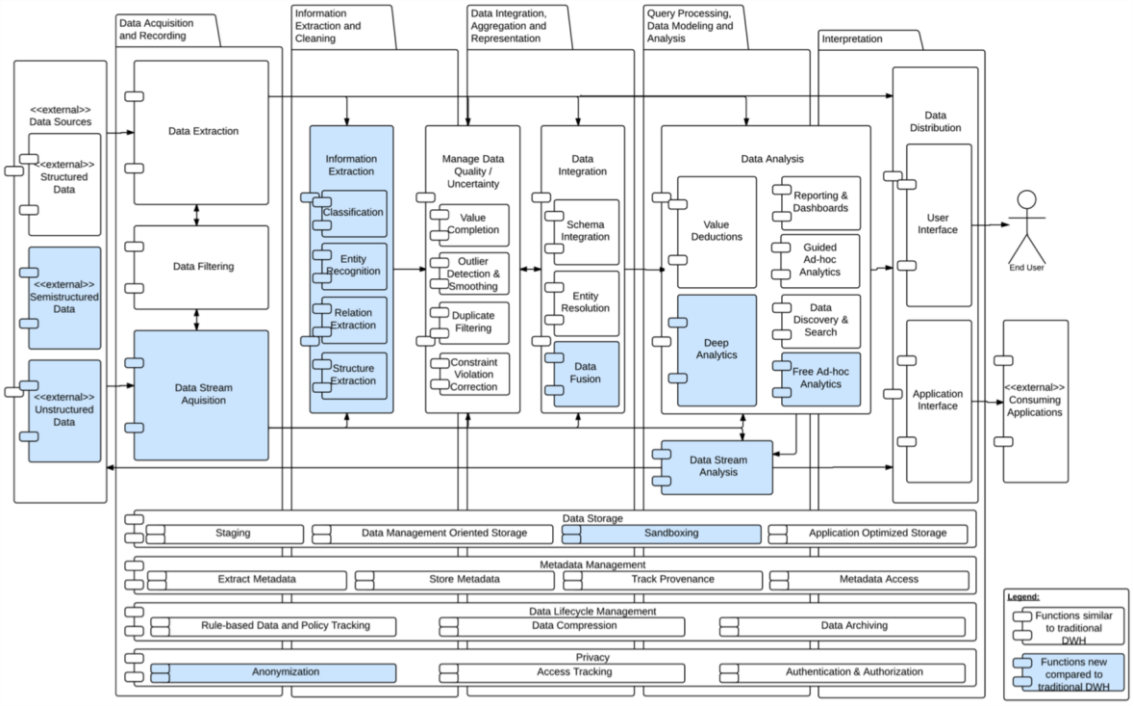

Figure 14. Meta Reference Architecture (Markus Maier, 2013)

Still, The Big Data landscape below (Figure 15) provides a sharp view of the most relevant technologies at one moment.

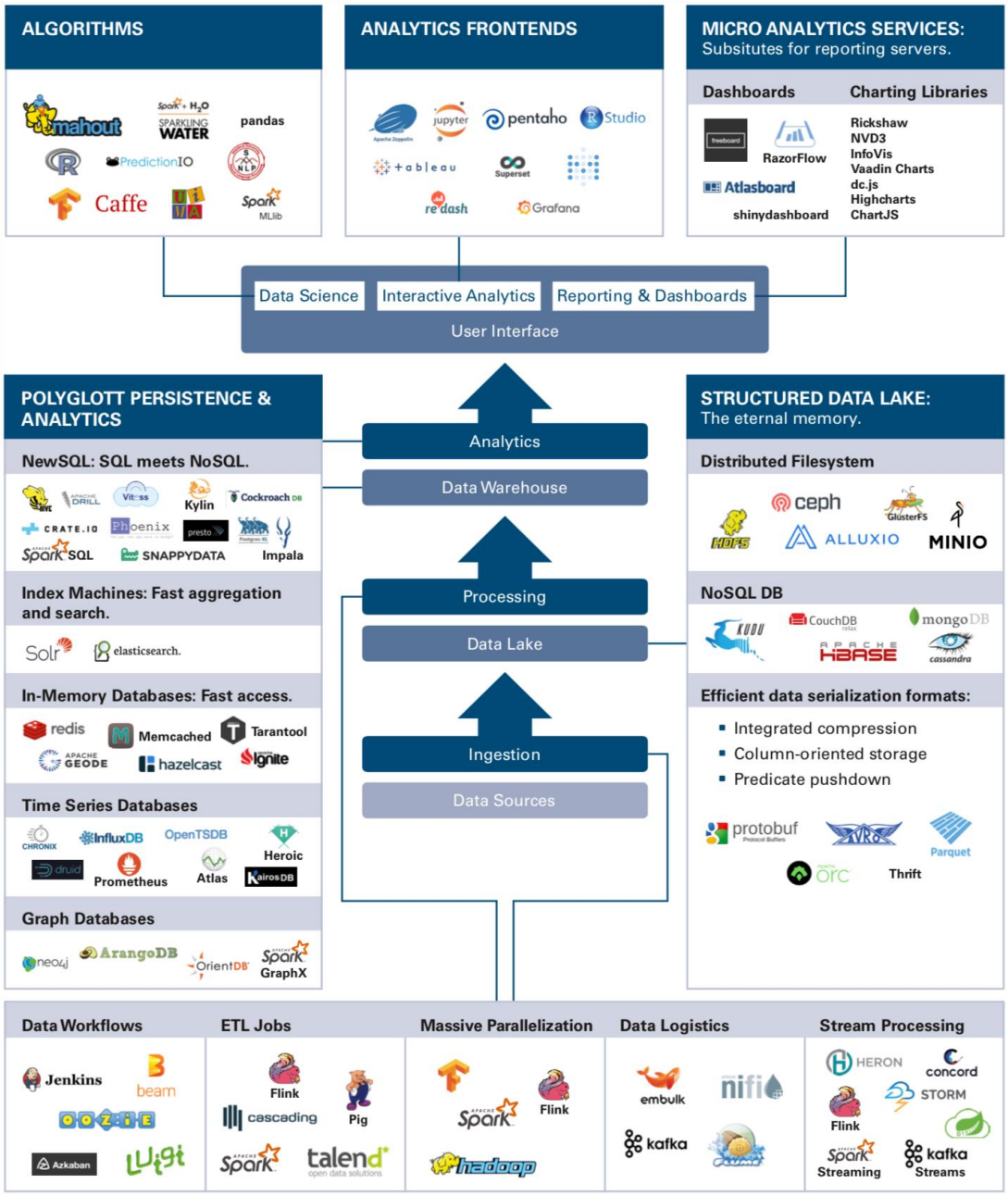

Figure 15. Organized Big Data Products 2019 (https://github.com/qaware/big-data-landscape)

4 APACHE SPARK

4.1 Introduction

Apache Spark is a fast and general engine for large-scale data processing (Apache Spark, Unified Analytics Engine for Big Data).

Apache Spark is an open source cluster framework initially developed in the AMPLab of the University of Berkeley, now under the umbrella of the Apache Software Foundation. Unlike Hadoop's MapReduce paradigm, spark primitives provide up to 100 times faster performance because of the use of memory buffer for intermediate results. However, pure performance on disk is already ten times faster than classical Hadoop MapReduce jobs.

In 2012, Matei Zaharia created the Spark project in the course of his Ph.D. studies. Spark was a response to limitations of MapReduce, which with its linear processing of the data flow, cannot handle efficiently the different workloads needed in machine learning, graph analysis, near real-time streaming. (Matei Zaharia, 2014).

“In 2015, *Spark project has more than 1000 contributors and is now one of the most active open source projects*” (Hui-Huang Hsu, 2017). Nowadays, it has over 1500 contributors, 1000+ companies (Openhub, Apache) and can run clusters up to 8000 nodes (Matei Zaharia, Spark Summit, 2014)

Apache Spark is:

- Fast because its advanced data flow execution engine (like a query execution optimizer in a database) and its in-memory computing (can use the disk as well); resulting in speed "*100x faster than Hadoop MapReduce in memory, or 10x faster on disk*” (Dmitry Timofeev, 2015).
- Polyglot. The APIs are available in Python, Java, Scala, R and provide SQL access.

- General. Spark is commode because it combines SQL, streaming, and sophisticated analytics like machine learning, graph processing, and optimization tasks.

4.2 Fundamentals

The developer writes an application in Spark in terms of operations on distributed data. The principal (and historically the first) building block is the Resilient Distributed Dataset (RDD).

An RDD is:

- A collection of objects spread across a cluster (JAVA objects)
- Immutable, so it cannot change once created
- Built through parallel operations (filter, add a column)
- Automatically recovery on failure
- Able to intermediary cache results in memory

An RDD is an immutable object once create; operations are like a recipe to create a new one.

There are two kinds of operations on an RDD:

- Transformations which are lazy activities to build RDDs from other RDDs
- Actions that return a result to store on a data sink (storage)

Spark optimizes the physical plan of execution like a database does for a query. It uses lazy evaluation, which is a technique that queues all transformations until the code wants to execute an action. This approach allows in-depth optimization of the execution flow.

Besides the RDD, Spark has several other APIs on top of which the most elegant and efficient is the DataFrame.

4.3 Architecture

Spark can work both in a single node and a cluster. In the general case (Figure 16), there are several running processes for each Spark application: a driver and multiple executors. The driver is the one who manages the execution of a Spark program, deciding the executor processes to perform the tasks. In the main program of a Spark application (the driver program), there is an object of type SparkContext, whose instance communicates with the cluster resource manager to request a set of resources (RAM, core) for executors.

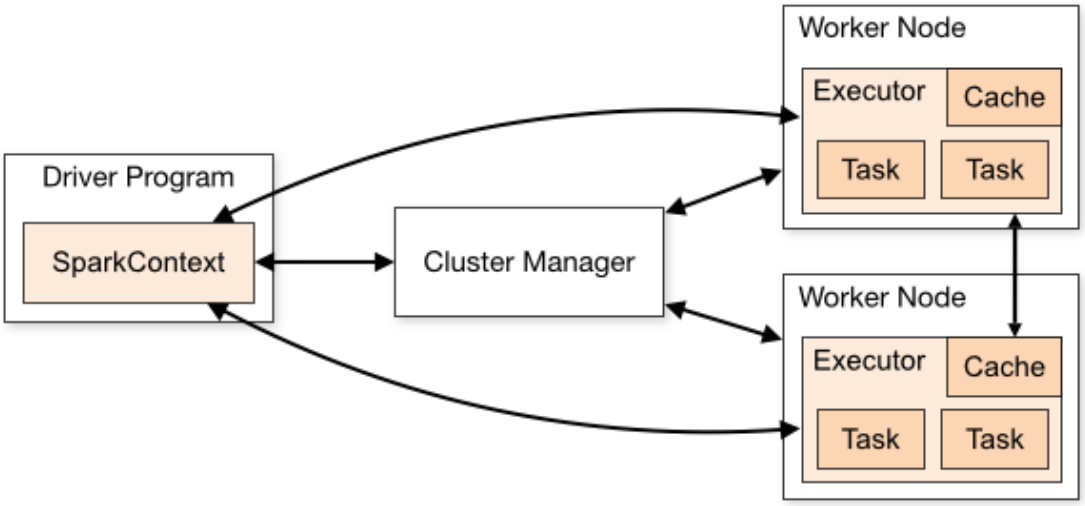

Figure 16. Spark Cluster View
(https://spark.apache.org/docs/latest/cluster-overview.html)

The system supports several cluster managers:

- A local and standalone, useful for development purpose.
- Apache Mesos, a general cluster manager.
- Hadoop YARN, the resource manager of the Hadoop software stack.
- Kubernetes, a general cluster manager for containerized applications.

4.4 Spark SQL and DataFrame

Spark SQL is a module for working with data structured as a sequence of identical lines (RDD allows the lines to be different). The code can mix relational and procedural operations, permitting complex analyzes through the DataFrame API. The DataFrame execution uses a highly efficient query optimizer, Catalyst.

4.5 Spark Streaming

Spark Streaming (Figure 17) is a component for the processing of data streams. Spark Streaming periodically creates a mini-batch containing the streaming input data and transforms stream processing into a sequence of Spark batch jobs.

Figure 17. Spark Streaming (http://spark.apache.org/docs/latest/streaming-programming-guide.html)

Spark Streaming provides the object DStream (Figure 18) for manipulating data streams, but today Structured Streaming is a more modern and the latest API to handling data streams.

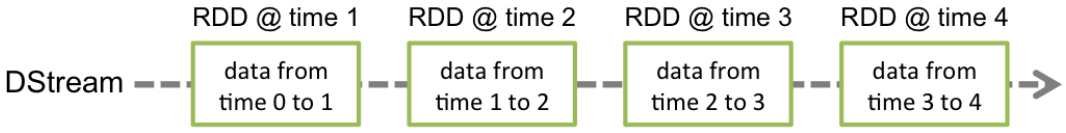

Figure 18. Spark DStream Micro-batches (http://spark.apache.org/docs/latest/streaming-programming-guide.html)

Structured Streaming filled the gap with a higher API just like the DataFrame API. As of SPARK 2.0, there is a promise that code can work with little maintenance in streaming and batch. "*The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive*" (Apache Spark, Structured Streaming Programming Guide).

In the example below (Figure 19), continuous counting of words is possible because SPARK maintains an unbounded table, on which the Spark Structured Streaming engine appends the new data coming from the stream.

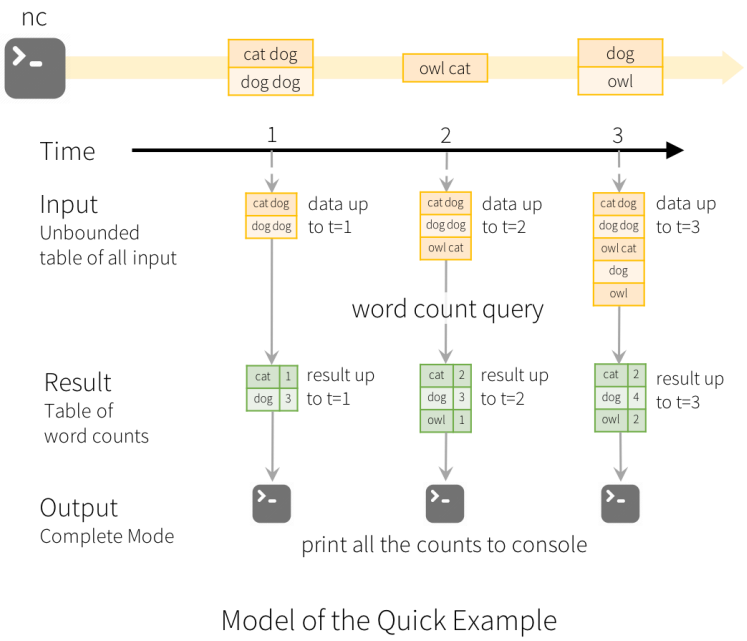

Figure 19. Spark Counting Word with Structured Streaming (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html)

There is a certain conceptual elegance to the unbounded table that contains all data past and present. The system achieves this by storing intermediate states. However, the API requires enough information that practically the unbounded table can be restricted to keep only the necessary data. *"Note that Structured Streaming does not materialize the entire table. It reads the latest available data from the streaming data source, processes it incrementally to update the result, and then discards the source data. It only keeps around the minimal intermediate state data as required to update the result."* (Apache Spark, Structured Streaming Programming Guide)

5 ARCHITECTURE DESIGN AND SOLUTION

5.1 Introduction

The key idea is doing some big data magic in a real-time way (Figure 20). The input is the web server access log, which is a semi-structured file, and the magic must handle batch and stream ingestion at scale. Out, something must show some analytics.

Figure 20. Architecture First Naïve Idea (Input, Processing, Output)

Despite not knowing the implementation path, one can already define some favorite options. The journey started with a set of preferences:

- No coding is better than coding
- Configuration over coding
- Declarative over procedural language
- SQL ready with window functions.
- Infrastructure must be fast to build/test (iterations).

The iterations through a final architecture followed a simple, lazy scheme. These questions can summarize the decision-making process:

- What is easy to install, to configure, and to test?
- What is general enough to solve more complex requirements?
- What to remove?

The items below (Docker, Python) helped substantially being agile and learning quickly.

5.1.1 Docker

Docker was central in the capacity to be agile and experimental in the quest of a Big Data infrastructure. Docker defines and runs artifacts called containers that are very similar to virtual machines but use resources isolation instead of virtualization (Figure 21).

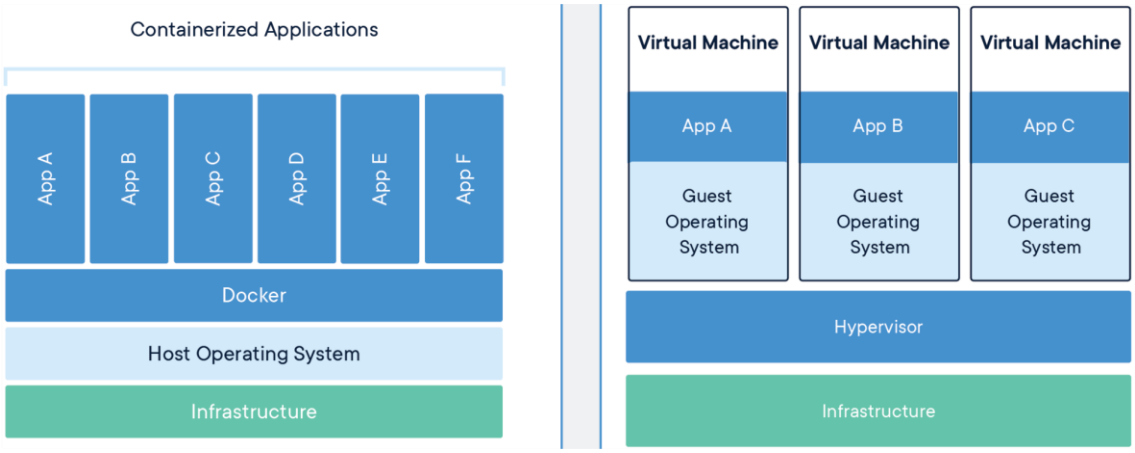

Figure 21. Comparing Containers and Virtual Machines
(Docker, What is a Container)

"A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings." (Docker, What is a Container)

The software application and the dependencies are tightly packaged together through the use of a recipe file, the Dockerfile. The final user needs not to worry about installing specific software with specific versions, possibly crashing the system.

5.1.2 Python

A goal was the coding effort minimization. However, this thesis author learned a new language because of the use of the massively parallel application framework

Apache Spark. Spark is polyglot regarding the programing language: Scala, Python, R, and Java. The choice of Python was straightforward since the programming language has exploded in popularity over the last few years, and it is easy to learn. It can do many things, basic scripts, basic and advanced statistics, and machine learning.

5.2 Architecture Experiments

5.2.1 The Search Engine Architecture

ELK is a favorite software stack to efficiently manage a large amount of data on which a process wants to search. The company Elastic (https://www.elastic.co/about/) delivers handy Docker documentation, ensuring that the developer can enjoy many functions fast. The stack strength is the interoperability of the several programs that make it.

Figure 22 below shows the first data pipeline implementation with ELK. Filebeat monitor the logs for recent appended information, then the tailed information is sent to Logstash for decoding and enrichment. Subsequently, these inputs are output to Elasticsearch, which takes care of indexing the data. Finally, Kibana provides a web user interface that simplifies searches within Elasticsearch indexes, allows data aggregation and a display under various out of the box diagrams and charts.

Figure 22. ELK Architecture

Components Descriptions

LogStash

LogStash is a log parser and open source event manager developed mainly because of the total lack of a defined standard for writing files Log. Its peculiarity

lies in the possibility of configuring it by using a simple text file. The data processed by LogStash follow a well-defined flow divided into three parts: Input, Filter, and Output. It starts by defining the input plugins that represent the various sources from which to draw data. Next, the events are filtered and parsed. Finally, the events, filtered or not, go to the output section, which is to direct the various events to the defined outputs.

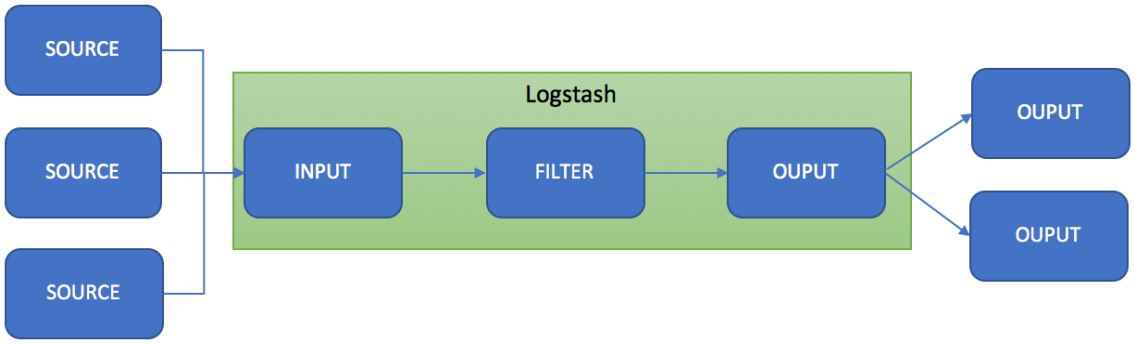

Figure 23. Logstash

Logstash makes available to developers, through Ruby language, the ability to create and use custom plugins. However, it has already over 80 standard plugins.

Elasticsearch

Elasticsearch is a document indexing and search engine based on Lucene with full-text capabilities and support for distributed architectures (Lucene is an open source API for information retrieval). The documents managed by Elasticsearch are in JSON format, which is a format suitable for the interchange of data between client-server applications. It subdivides indexes into fragments or shards.

Kibana

Kibana is an open source platform designed to interact with Elasticsearch. With Kibana it is possible to search and view the data stored by Elasticsearch through queries made available by a very intuitive web graphic interface. The program allows to aggregate the data present in the chosen Elasticsearch cluster and to visualize different charts (pie, column). Searches are done efficiently (like a web search) for a specific indexed field, such as the IP client in a weblog, and return a list of the documents where there is an occurrence.

Conclusions

The ELK stack has excellent potential and is often upgraded. It is worth noting:

- That it is entirely free. However, a commercial featured pack exists and brings enterprise security integration and advanced display for time-series.
- The availability of Docker images and extended documentation
- The real-time monitoring features in Kibana.
- It provides compelling tools to perform full-text and real-time analytics.

In terms of cons:

- Elasticsearch automatically guesses the field types when receiving its first document: it is schemaless. However, it is not 100% accurate, and therefore the schema needs to be specified. The task is tedious because Elasticsearch schema can contain hundreds of fields.
- Modeling data with the document concept is missing extensive academic or practitioner reference literature.
- Elasticsearch's query DSL is not SQL (although recently added has a commercial feature)
- It can retrieve a few to thousands of records but not suited to massive data processing.

Conclusion: ELK stack is not appropriate for intense data processing. The author kept it as user serving layer and added Apache Spark as a parallel data processing engine.

5.2.2 Toward A Lambda Architecture

This iteration tests the batch layer of a Lambda architecture (Figure 24). Usually, different environments implement the batch and the real-time layer, which also duplicate the code in two places. Apache Spark stack promises a lambda architecture compliant system with a unified development that supports both batch and streaming operations. In principle (the conclusion chapter is going to say that is not true), with little effort, the batch and streaming code are interchangeable.

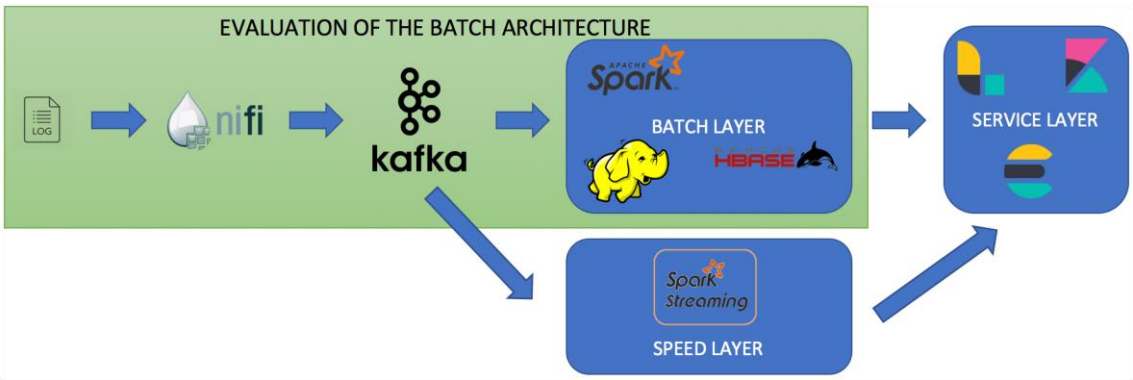

Figure 24. Batch architecture

Components Descriptions

Kafka

Apache Kafka is a distributed and high-performance open source message broker based on the writing of logs. Each operation adds a message to the queue, technically an append of a line in a distributed log file. Kafka presents a publish-subscribe architecture that allows the subdivision of messages into topics, the parallelization of readings and writes on several machines, the management, and deletion of messages based on a key or a timeout. Besides this, it offers redundancy through a system of replicas so that the content is always available even after failures.

NiFi

Apache NiFi is an open-source software developed by the NSA. It uses components called processors, which allow operations such as capturing data from sources, transforming and publishing the data flow on external systems. To date, in the framework, there are over 200 different processors available. A processor usually has three outputs:

- Failure: If the data flow cannot be processed correctly, it routes the original data to this output.
- Original: It routes the original data to this output.
- Success: It routes successful data to this.

Key aspects of this framework are:

- real-time control to manage the movement of data between multiple systems;
- the possibility of working in cluster mode, guaranteeing properties such as scalability, fault tolerance, and availability
- source of data traceability,
- a web-based user interface,
- back pressure support and buffering essential for managing different frequencies of data arrival within the system.

NiFi can be used for a simple event processing, as it allows simple operations on data. However, it does not enable complex event processing. It integrates well with Kafka through the appropriate processors.

HBase

HBase stores the results obtained from the processing phase. Apache HBase is a distributed, scalable, NoSQL database that runs on a Hadoop cluster. It can host huge tables with billions of rows and millions of columns. Random read/write access is possible.

Conclusions

NiFi UI is a real advantage over configuration file-based tools (such as Logstash). Data-flow programming is done visually by assembling boxes and arrows, writing zero lines of code. However, it is over-engineered for the routing of a simple web log file. NiFi requires a load balancer (HAProxy), a synchronization service (Apache Zookeeper), and a service to share the dataflow design (NiFi-Registry).

HBase is efficient when inserting new data and allows relatively low latency on random data access if the query uses the full key. It is a schemaless NoSQL database, so data schema design is very flexible (two rows can have different schemas). The lack of an SQL interface is a minus. Also, it is not trivial to get even a small fully distributed HBase running. As it is a master-slave architecture built on top of the distributed file system HDFS, itself a master-slave architecture, the setup includes many components.

HBase is not a ‘self-sufficient' technology for data storage and management; using a Hadoop cluster is necessary. Apache Ambari used in Docker containers made the full setup easier. Although this method works, a conventional container approach would avoid installing too much software inside a container and rely more on an original recipe (Dockerfile). The initial Hadoop framework was historically fundamental and still very important for on-premise installation, but it is also notoriously difficult to get it right (Alex Woodie, 2017, 2019).

5.2.3 Toward A Kappa Architecture

This step assessed the Kappa architecture (Figure 25), which is a simple streaming solution. Kafka or any messaging/queueing system is a significant component of such architecture, and as it also has its place in a batch-oriented system, it has been part of the testing from the beginning. It also introduced a data lake to simplify the previous iteration made with Hadoop/HBase. There was no focus on the Service layers except the intention of challenging it in the next iteration. This trial is about the streaming module of Apache Spark.

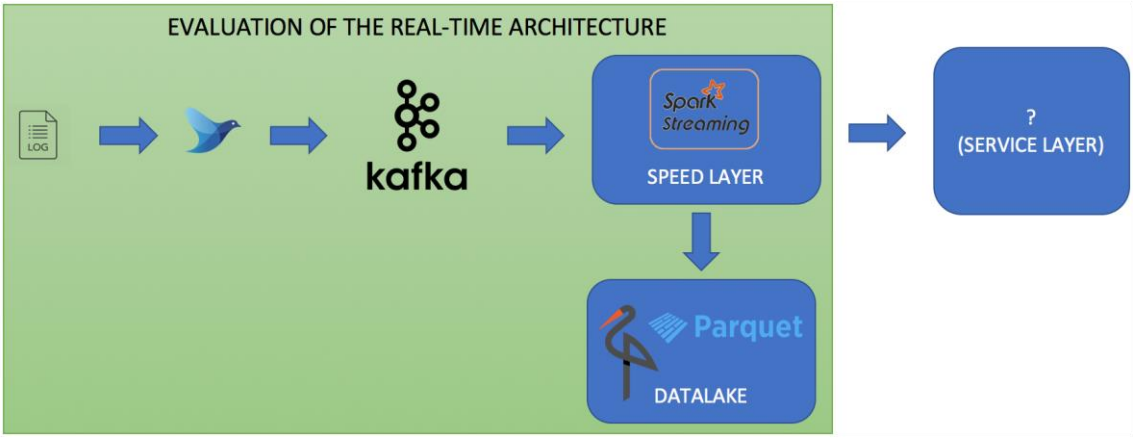

Figure 25. Kappa Streaming Architecture

Components Descriptions

There is a description of Apache Spark in §5.5 and Parquet in §3.5.1.

MinIO

MinIO is an object storage server compatible with Amazon S3 cloud storage service. It can store files from a few KBs to a maximum of 5TB. MinIO server is light in terms of memory and CPU used and requires little configuration.

Conclusions

MinIO and Parquet worked like a charm. On the contrary, Apache Spark Structured Streaming turns out to be cumbersome because of a few limitations. The streaming engine cannot use the previous sessionization code because it relies heavily on SQL analytical functions (LAG and SUM over a partition). However, it was too late to learn a new data processing framework, and also the author appreciated the non-structured streaming version of Spark very much.

5.2.4 Final Architecture

The final architecture (Figure 26), unlike the classical Hadoop model, separates the computation from the data storage. It generalizes the usage of the data lake (§3.3.3) as the central data store which stores the raw data but also the final results such as the aggregated data. Those data are once written in data lake accessible to any Business Intelligence (§3.4) tool that can connect to the distributed SQL engine Presto such for example Tableau or Microsoft PowerBI. Jupyter is for simple to sophisticated data analysis and can use a Spark cluster for getting the result.

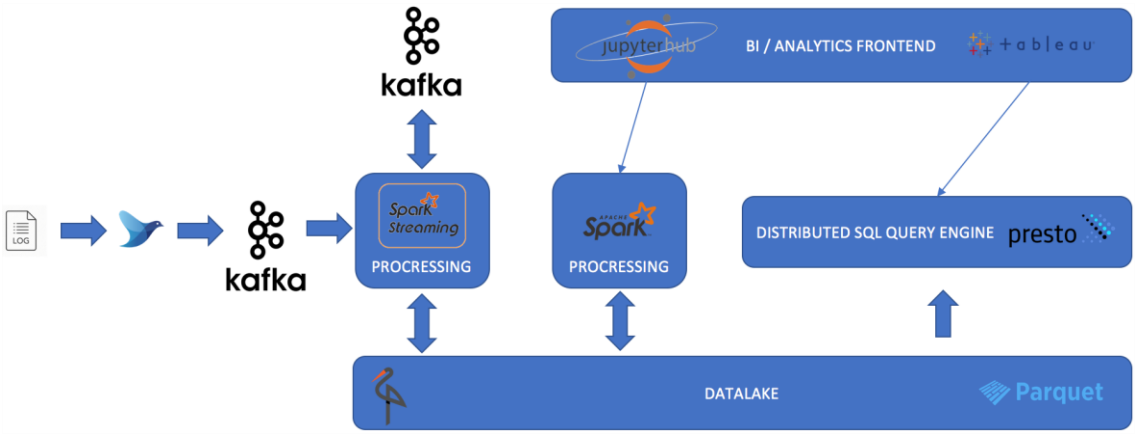

Figure 26. Final Architecture

There is a second usage of Kafka as Spark Structured Streaming requires a re-factoring of the sessionization code on a sequence of smaller queries. Here, Kafka is, in fact, a temporary queue/store.

Components Descriptions

Presto

Presto is an Apache open-source SQL query engine designed to handle data warehousing and analytics: data analysis, aggregating massive amounts of data and producing reports. With this component, a query goes where the data live in the storage system. A single Presto query can combine data from multiple sources including standard data lake solutions, databases or even message brokers such as Kafka. The targeted response time ranges from sub-second to minutes, and data sources size varies from gigabyte to petabyte. Presto works with standard BI tools such as Tableau, PowerBI, MicroStrategy, QlikView, and Excel, and any program using JDBC.

JupyterHub

JupyterHub is a multi-user version of the Jupyter notebook server. It is a web-based interface useful for rapid prototyping of data-related projects. A Jupyter notebook has a read–eval–print loop (REPL) approach that makes the coding very interactive. The notebook also contains visualizations and narrative texts.

Conclusions

With this last iteration comes it remains little of Hadoop. The setup is smooth, light and still gives powerful capabilities. The only pain point is in the restrictions of Apache Spark Structured Streaming. These limitations make the code a magnitude more complicated, but the near real-time capacity of Spark is actual. Below, §5.3 explains the limitations and workaround5.3.

5.3 Solution - Architectural View Model

This paragraph aims to give complementary views on the software developed during this thesis.

5.3.1 Functional view

The functions depicted below (Figure 27) are the core of the clickstream implementation. From this point of view, batch or streaming iterations are precisely the same.

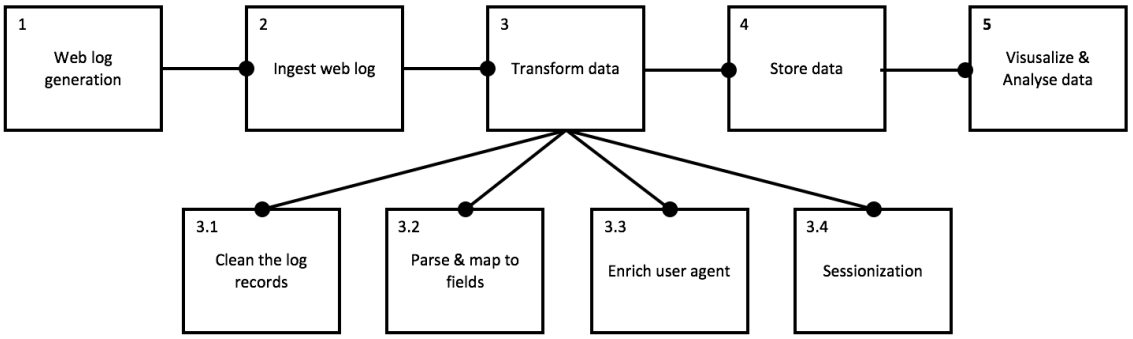

Figure 27. Functional View of the Data Pipeline

The application is a data pipeline, and it is a pure sequence of functional transformation with the following items:

1. A generator of click based on the real website logs. It adds the click records on the weblog, at the same time and with the same cadence as the original file, except for the date that the program actualizes.
2. A file tail program sends the weblog lines every minute to an ingestion engine (Apache Kafka).
3. A logic transforms the weblog lines
   3.1. Cleaning: a code skips IIS log header lines. (no other cleaning)
   3.2. The log lines are a semi-structured text encapsulated in JSON when extracted from Kafka. So, a process decodes JSON, then parse the web logline, and maps the expected fields in a table like structure.
   3.3. User-agent strings are parsed and enriched with metadata such as the operating system of the user, the device type (desktop, laptop, smartphone).
   3.4. Logic computes sessionization

4. Then the result is stored in the data lake.
5. Presto or Jupyter can finally access data lake stored data.

5.3.2 Process view

Most of the functions are very trivial except sessionization. The implementation followed the industry standard of 30 min (§2.4) with the last click to count a new user session. The table below shows a way to solve it.

Table 2. An example of Sessionization

USER ID	TIME	LAG	DIFF	> 30?	CUMSUM	SESSION_ID
21	12.00		0	0	0	21_0
21	12.01	12.00	1	0	0	21_0
21	12.20	12.01	19	0	0	21_0
21	12.59	12.20	39	1	1	21_1
21	13.05	12.59	6	0	1	21_1
21	13.40	13.05	35	1	2	21_2
21	13.45	13.40	5	0	2	21_2

A row compares the time with the previous row, which gives the time since the last click. If this is higher than the 30 min threshold, it is a new session. A cumulative sum gives the session number. Concatenation with user ID gives a unique session ID. This computation is natural in SQL with the help of the analytic function LAG() and SUM() and the window operator PARTITION BY.

Unfortunately, those analytic functions are not in Spark Structured Streaming (it is unknown when they will be available). Still, it is possible to implement the sessionization by using Scala instead of Python and with the procedural API flatMapGroupsWithState. The sad part is that it requires Scala, and to develop with a robust but low-level API. Too bad to develop with a low-level API when SQL is so useful, and probably part of the success of Spark.

To stay, with Python and with a DataFrame/SQL API, the code must implement the sessionization in with left outer joins and classical group by instead of the powerful LAG and SUM of modern SQL. Joins usually are not optimum, but at least it worked, and Spark SQL optimizer did a great job. Hopefully, the data were small and user sessions short on the website.

Another limit is that only one aggregation is possible at a time in Structured Streaming. Inevitably, the sessionization became several pieces of code running in several Spark Streaming processes. Figure 28 below shows how the micro sub-queries are run to avoid the limitation. The sessionization uses Kafka as temporary streaming storage.

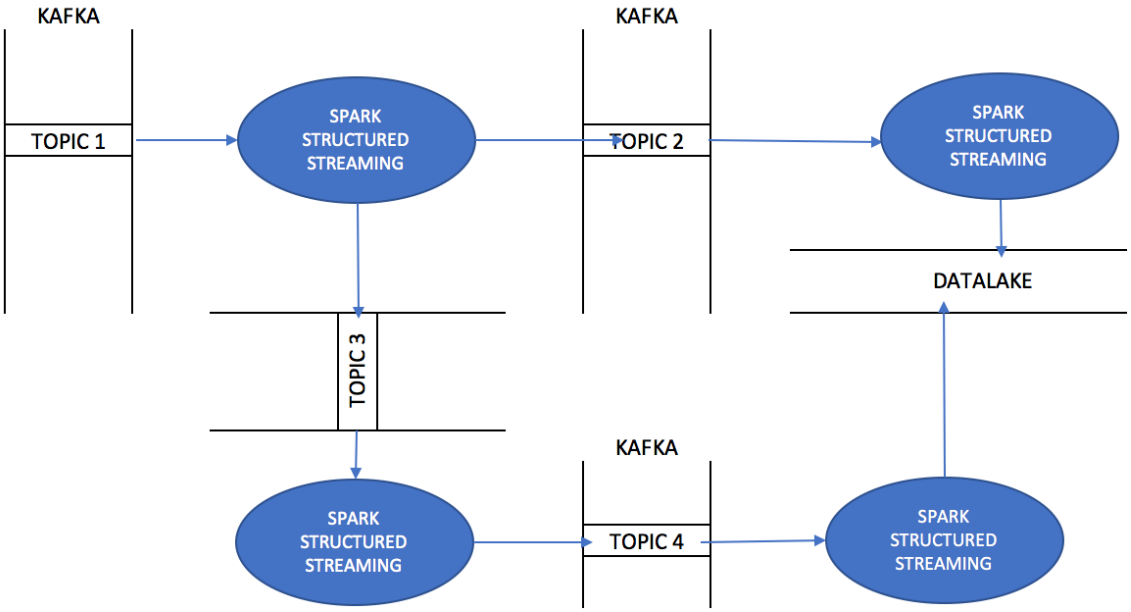

Figure 28. A process View (Continuous Micro-Queries)

The picture below (Figure 29) shows the impossibility of giving precise session information before the beginning of the next one.

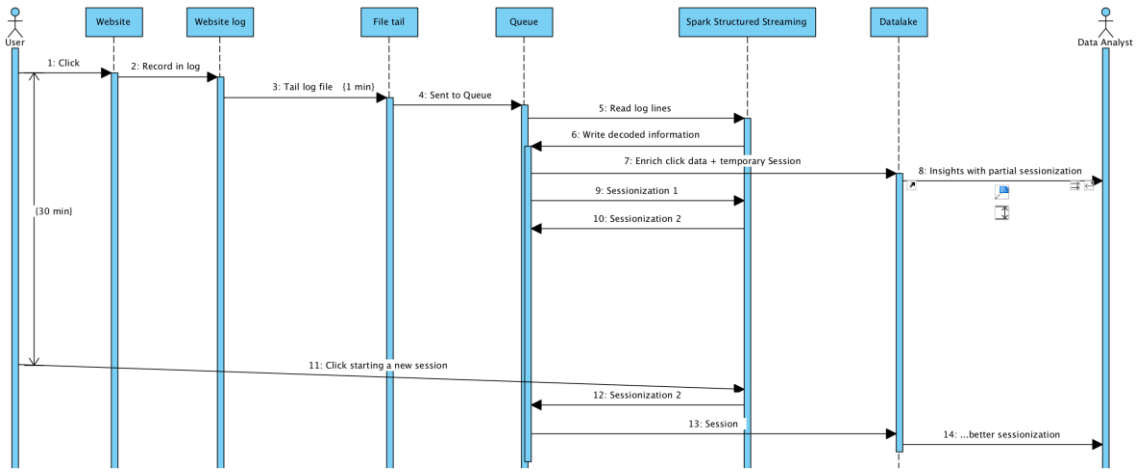

Figure 29. A Sequence View of the Sessionization

The more the web sessions are long, the more the system keeps data in the unbounded table (see §4.5) to compute the sessionization (Figure 30). The streaming solution must keep the unbounded table limited in size, which limits the precision of the sessionization.

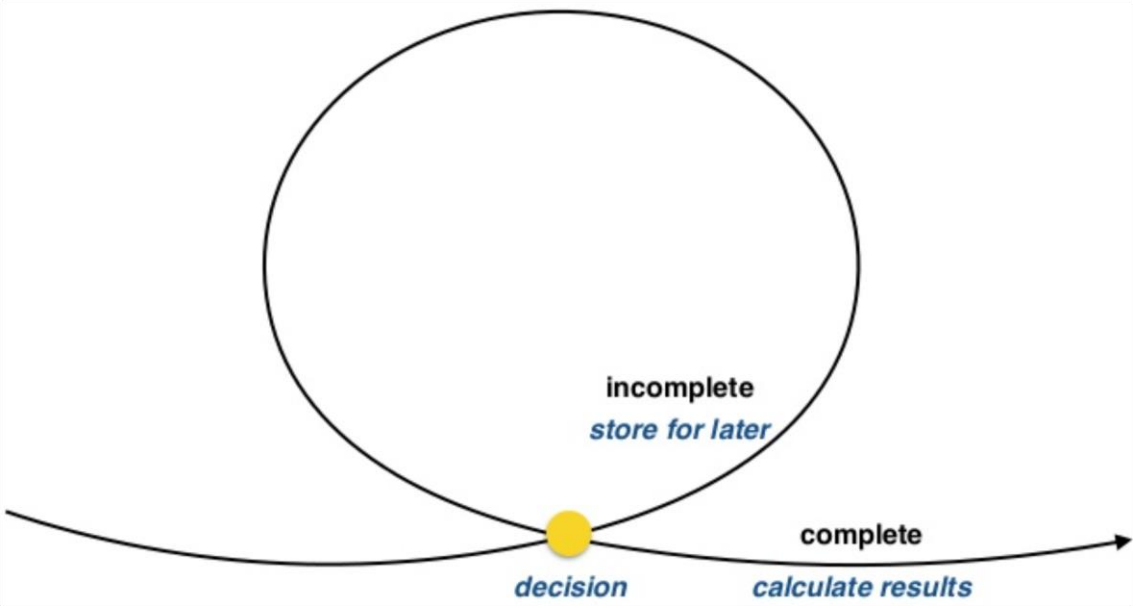

Figure 30. Incomplete Sessions / Unbounded Table
https://www.slideshare.net/RamnasUrbonas/sessionization-with-spark-streaming

A very long session is an indicator that the user may be a program (a bot) than a human. If possible, memory savvy approach recommends filtering out known bots such as Google or Yahoo index services.

5.3.3 Physical view

The solution ran on a single host but with distributed components. Something possible to do with Docker. Below is Figure 31 with the different Docker hosts involved. Not cited in the rest of the thesis, but Kafka in cluster mode required the help of a triumvirate of Zookeeper processes. Presto requires a Hive repository to run because it saves metadata information there.

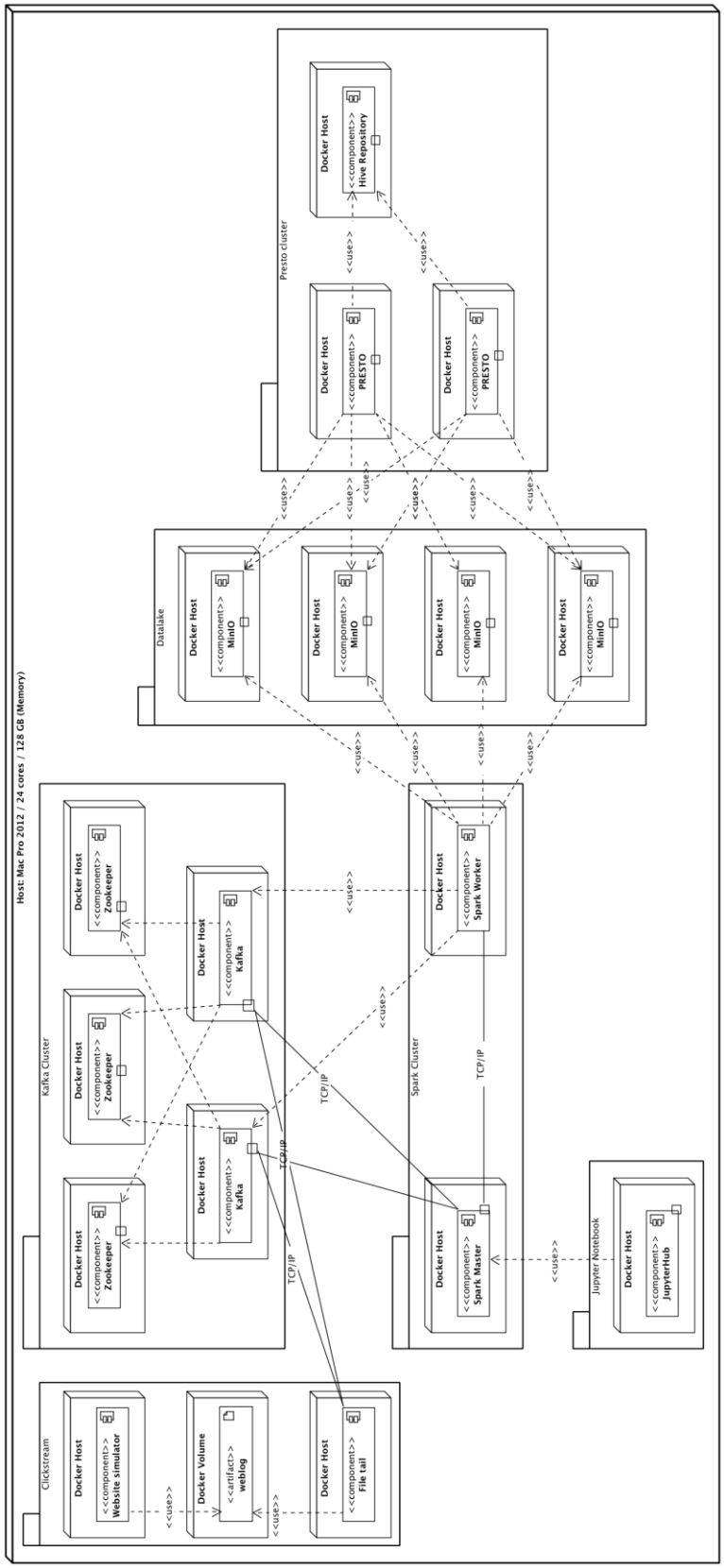

Figure 31. A Physical View of the Docker Hosts

6 DISCUSSION

Web usage mining has been since the Internet become mainstream (end of the '90s), the cornerstone of adapting web sites to increase the profits (Chapter 1). The clickstream analysis (Chapter 2) is a sub-topic of the web usage data mining, which focuses on the records generated while a user clicks on the parts of his/her browser screen. Those records are typically semi-structured website log files growing with the Internet expansion to become gigantic. With web success, even the growth rate is speeding up. Eventually, analyzing clickstream data always had those V characteristics (§3.2), nowadays said to be the characteristics of Big Data: variety, volume, velocity, and value.

The phenomenon of Big Data has by now taken over. There is a constant challenge to look for and optimize, always new analysis processes on the information, accumulated for years and years. Organizations quickly assess data that seemed to have no real importance to become protagonists in this new era of data management (Chapter 3). To optimize the processes and increase the profits, the analysis of streaming data allows companies to take the best business strategies in a short time, as soon as a change is occurring. It makes sense, for example, that a security breach gets an appropriate answer within seconds (§1.2).

Clickstream analysis is a mature field with hundreds of researches in the last 25 years. The hype behind it has long vanished, and Google Analytics has the crown with 58,85% of the market share (Datanyze, Web Analytics Market Share Report). An initial idea of this thesis is that clickstream analysis is a natural-born big data challenge. Thus, it can start a journey with less risk of having a subject that does not fit the Big Data world. The subject also fits the streaming context (e.g., Amazon recommendations are covered by real-time streaming and by batch applications).

This thesis elected three clickstream functions to implement: data cleansing, user sessionization (§2.4) and user agent enrichment. The analysis and the implementation run interactively, iterating with, as little setup as possible (Chapter 4), rather than theoretically comparing and motivating with different arguments the choice

of frameworks for each function. In particular, sessionization and the easiness of its implementation became a benchmark to compare the various iterations. Conversely, the heaviness of the different infrastructure components configurations was a reliable indicator too.

The practical part of this thesis tested a series of open-source software solutions for the development of a Big Data architecture (§3.3) that would allow the following capabilities (§3.6):

- Ingestion pipeline for consuming the click records
- Transformations for data cleansing, user sessionization, enrichment (user agent enrichment)
- Storage of data
- Analytics &
- Visualization for presenting accessible insights

Though the beginning was technically agnostic by necessity (not enough knowledge of the Big Data field), soon a preference rose around an initial set of tools:

- Spark (Chapter 5) is a more advanced product than Hadoop. It is newer and up to 100 times faster. Also, it has been one of the most active of all open source Big Data applications (The Apache Software Foundation Blog, 2014). Spark Streaming is another element of the framework (§5.5), which allows applications to perform analytics on streaming, real-time data. A disappointing factor is that Spark promises that the logic and the code base for batch and streaming would be almost the same. In fact, Spark Streaming, precisely Spark Structured Streaming does not support the SQL analytical functions (operator such as Lead, Lag, Sum), plus other limitations with SQL joins. The author redesigned the code of the sessionization, previously developed for Spark batch, to continue with Spark Structured Streaming and the Python Language. However, even with Scala, the SQL limitations would have remained.
- Because of the costs, to get flexibility and hands-on, the developments were on a local host instead of a cloud environment such as Amazon or Azure. The machine chosen is a Mac Pro 2012 with 128 GB memory and 24 virtual cores. All the components for even a simple Big Data solution

are impossible to configure on the same host because of software library conflicts and network port issues. Therefore, a unique host requires some level of isolation. To achieve that isolation, Docker containers (§4.1.1) were a more versatile and light approach than setting up virtual machines. The gain was noticeable for the memory as Docker does not duplicate the operating systems. However, also the recipes approach of Docker allows high control on what is running inside the containers.

- For coding with Spark, programming languages are Scala, R, and Python. All languages are very remarkable, but the balance swung in favor of Python (§4.1.2) because of its enthusiastic adoption by programmers working with Big Data and data science makes it a good investment.

6.1 Achievements

The implementation was using multiple trials moving step by step through different options (Chapter 4).

A first realization (§4.2.1) based on the software stack around Elasticsearch provided a knowledge of the weblog data. With zero coding, it was possible to ingest the weblog in streaming, clean and enrich the data with user agent information, store the data and visualize with high-quality dashboard/charts. User sessionization, supposedly achievable, would have required to invest much time in the proprietary request language of Elasticsearch. A need for a more general and massively parallel date engine emerged.

A second phase (§4.2.2) introduced more powerful tools: Spark for transforming the data, NiFi for ingestion, and Kafka as a message broker. Processing the data in a batch made easier the elaboration and the computing the sessionization with Spark. This step confirmed Spark as the central component of this thesis. NiFi was an over-engineering regarding the simplicity of accessing a weblog, and HBase based on Hadoop is a too complicated infrastructure component.

The main themes of the third implementation (§4.2.3) were the simplification of the data storage and a first glimpse of the Spark Streaming features. Hadoop clustering was entirely removed to the profit of a data lake approach (§3.3.3).

MinIO an Amazon S3 compatible filesystem was an easy to configure data lake layer, and Spark was reading and writing Parquet file on it (§3.5). Visualization was not part of the test but raised the idea, that using Elasticsearch only for displaying data under use the potential of the duo data lake/Parquet. (reference BI)

The last experiment (§4.2.4) showed that it is possible to process streams of clickstream data, coming continuously from a weblog, with low latency by using Spark Structured Streaming. However, the last version (still valid in 2.4) of Spark Structured Streaming has so much SQL limitations that the sessionization code developed during the batch trial (second test) became four different pieces code glued together by Kafka acting as a temporary/transient data store. With the Scala API, it would have been possible to develop the sessionization with only a piece of code (still not the SQL interface), but not with Spark Python API (the choice of this thesis) which is more limited. On the side of the visualization, data stored as Parquet files, the data lake (MinIO) and Presto (a distributed SQL engine) made easy the connection of and the data retrieval from, Business Intelligence tools such as Tableau. Also, Jupyter notebook system allowed more advanced and interactive data queries (It uses the Spark cluster directly to get the data).

6.2 Future directions

This thesis, exploratory by nature, crossed many technologies and frameworks that could set new directions or bring improvements:

- Julia: a new language (2012) a dynamic and straightforward syntax like Python but which is much faster, mathematical computation oriented and with built-in parallel processing features (Julia website, Micro Benchmarks). Jupyter has a Julia kernel and therefore allows ad hoc analysis with it.
- Kubernetes: a container platform. The implementations were sets of Docker containers. A natural extension would be an orchestration tool such as Kubernetes or Apache Mesos. Those tools add advanced scheduling, deployment tools, services discovery, scaling and load balancing.

- Apache Kudu: A storage layer and column storage format that has comparable performance with Parquet. It permits fast row updates where Parquet is an append file scheme. It can bring different data modeling of the result files.
- Alluxio: A virtual distributed storage system. It is convenient to have an abstraction between the data processing components and the real storage system. We can imagine even using several storage systems. With this thesis, Alluxio can lie between Apache Spark or Presto and the storage supported by MinIO (Amazon S3 compatible).
- Neo4j: a graph database. Browsing one or several websites is a path in a graph; that is where a graph database permits a better design of the information model and provides a powerful graph query language.
- TimescaleDB: a time-series database. Clickstream records are irregular time-series where data can miss, where the spacing of observation times is not constant. Time-series database does an excellent job in terms of ingestion speed, efficient storage, and quick processing of requests such as re-sampling down or up (e.g., re-sampling millisecond data to second).

Spark Structured Streaming in the last version 2.4 is far to have reached the maturity of the non-streaming version. Some current limitations hopefully are temporary (e.g., missing SQL analytical functions or type of data joins allowed) and a such it is part of any future direction.

REFERENCES

Alex Woodie. Hadoop Has Failed Us, Tech Experts Say. Created on 13.03.2017. Printed on 28.04.2019. https://www.datanami.com/2017/03/13/hadoop-failed-us-tech-experts-say/

Alex Woodie. Here's What Doug Cutting Says Is Hadoop's Biggest Contribution. Created on 01.04.2019. Printed on 28.04.2019. https://www.datanami.com/2019/04/01/heres-what-doug-cutting-says-is-hadoops-biggest-contribution/

Apache Spark. Structured Streaming Programming Guide - Spark 2.2.3 Documentation. Printed on 28.04.2019. https://spark.apache.org/docs/2.2.3/structured-streaming-programming-guide.html

Apache Spark. Unified Analytics Engine for Big Data. Updated on 26.04.2019. 2019-04-26. Printed on 28.04.2019. http://spark.apache.org

Bernard Marr. IBM Big Data & Analytics Hub. Why only one of the 5 Vs of big data really matters. Created on 19.03.2015. Printed on 28.04.2019. https://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters

Chuck Densinger, Mark Gonzales. There's no such thing as unstructured data. Created on 05.09.2016. http://analytics-magazine.org/theres-no-such-thing-as-unstructured-data/

Databricks. Spark Summit July 2017. Printed on 28.04.2019. http://spark-summit.org/wp-content/uploads/2014/07/Sparks-Role-in-the-Big-Data-Ecosystem-Matei-Zaharia1.pdf

Datanyze. Web Analytics Market Share Report | Competitor Analysis | Google Analytics, Facebook Analytics, Google. Printed on 28.04.2019. https://www.datanyze.com/market-share/web-analytics

Dmitry Timofeev. Meetup#4, Apache Spark as SQL Engine. Created on 28.10.2015. Printed on 28.04.2019. https://www.slideshare.net/SPb_Data_Science/meetup4-apache-spark-as-sql-engine

Docker. What is a Container? Printed on 28.04.2019. https://www.docker.com/resources/what-container

Eric Brewer. Created on February 2012. Twelve Years Later: How the "Rules" Have Changed. Printed on 28.04.2019. http://alchem.usc.edu/courses-ee599/downloads/T_CO2_CAP12YearsLater.pdf

Google. How a web session is defined in Analytics. Printed on 28.04.2019. https://support.google.com/analytics/answer/2731565

Hui-Huang Hsu, Chuan-Yu Chang, Ching-Hsien Hsu, Morgan Kaufmann. 2017. Big Data Analytics for Sensor-Network Collected Intelligence.

J. Goossens, P. Richard. 2004. Overview of real-time scheduling problems, Euro Workshop on Project Management and Scheduling.

Jay Kreps. Questioning the Lambda Architecture. The Lambda Architecture has its merits, but alternatives are worth exploring. Created on 02.07.2014. Printed on 28.04.2019. https://www.oreilly.com/ideas/questioning-the-lambda-architecture

Julia. Micro-Benchmarks. Printed on 28.04.2019
https://julialang.org/benchmarks/

Markus Maier. 2013. Master's thesis. Towards a Big Data Reference Architecture.

Matei Zaharia. 2014. An Architecture for Fast and General Data Processing on Large Clusters. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.pdf

Nathan Marz, James Warren. 2015. Big Data: Principles and best practices of scalable realtime data systems.

Nathan Marz. How to beat the CAP theorem - thoughts from the red planet. Created on 13.10.2011. Printed on 28.04.2019. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

Osmar R. Zaiane, Jaideep Srivastava, Myra Spiliopou-lou, Brij Masand. 2002. WEBKDD 2002 - Mining Web Data for Discovering Usage Patterns and Profiles: 4th International Workshop, Edmonton, Canada, July 23, 2002, Revised Papers, Page 69/70.

Panopticlick. What is fingerprinting? What does it mean if my browser is unique? Printed on 28.04.2019. https://panopticlick.eff.org/about#browser-fingerprinting

Pekka Pääkkönen, Daniel Pakkala. 2015. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems
Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems

S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. 2011. Dremel: interactive analysis of web-scale datasets. Commun. ACM, 54(6):114–123.

The Apache Software Foundation Blog. The Apache Software Foundation Announces Apache™ Spark™ as a Top-Level Project. Created on 27.02.2014. Printed on 28.04.2019. https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50

Wikipedia. Click path. Updated on 22.03.2019. Printed on 28.04.2019. Wikipedia. https://en.wikipedia. org/wiki/Click_path

Wikipedia. Real-time computing. Updated on 19.04.2019. Printed on 28.04.2019. https://en.wikipedia.org/wiki/Real-time_computing

Wikipedia. YouTube. Updated on 28.04.219. Printed on 28.04.2019. https://en.wikipedia.org/wiki/YouTube

Word Wide Web Consortium (W3C). Web Characterization Terminology & Definitions Sheet. Updated on 02.10.2017. Printed on 28.04.2019. https://www.w3.org/1999/05/WCA-terms/01

Youtube. Press. Global Reach. Printed on 28.04.2019. https://www.youtube.com/yt/about/press/