Rafael Melo Coimbra

Plataforma Baseada na Arquitetura Lambda Aplicada a Cenário IoT

Framework Based on Lambda Architecture Applied to IoT Case Scenario

Rafael Melo Coimbra

Plataforma Baseada na Arquitetura Lambda Aplicada a Cenário IoT

Framework Based on Lambda Architecture Applied to IoT Case Scenario

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor Óscar Mortágua Pereira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Rui Luís Aguiar, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais, irmão, avó e à Lara

O júri / the jury

Prof. Doutor Aníbal Manuel de Oliveira Duarte

Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Presidente / presidente

Prof. Doutor Hélder José Rodrigues Gomes

Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro

Vogais / examiners comittee

Prof. Doutor Óscar Narciso Mortágua Pereira

Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Agradecimentos

Aproveito agora para agradecer a todas as pessoas que me ajudaram durante o meu percurso académico
Ao professor Doutor Óscar Mortágua Pereira, meu orientador, por toda a ajuda que me foi dando ao longo de todo o processo e;
Aos meus pais, pelo apoio incondicional que me deram fazendo com que todo este percurso fosse possível.

Palavras-chave Big Data, Processamento em tempo real, Processamento em bloco, Arquiteturas distribuídas e escaláveis, Arquitetura Lambda, NoSQL, Internet das Coisas, Data Analytics.

Resumo

Desde o início da primeira década do presente milénio, tem-se testemunhado um aumento exponencial da quantidade de dados produzidos de dia para dia. Numa primeira instância, o aumento foi atribuído aos dados gerados pelos dispositivos GPS; numa segunda fase, à rápida expansão das redes sociais, agora não devido a um fator específico, mas devido ao surgimento de um novo conceito denominado de Internet das Coisas.

Este novo conceito, com resultados já mensuráveis, nasceu da premissa de facilitar o dia-a-dia das pessoas fazendo com que os dispositivos eletrónicos comunicassem entre si com o objetivo de sugerir e assistir a pequenas decisões dado os comportamentos observados no passado.

Com o objetivo de manter o conceito possível e o estender para além das já existentes aplicações, os dados gerados pelos dispositivos necessitam não apenas de serem armazenados, mas igualmente processados. Adicionando ao volume de dados a sua variedade e velocidade de produção, estes são igualmente fatores que quando não ultrapassados da maneira correta podem apresentar diversas dificuldades, ao ponto de inviabilizarem a criação de novas aplicações baseadas neste novo conceito.

Os mecanismos e tecnologias existentes não acompanharam a evolução das novas necessidades, e para que o conceito possa evoluir, novas soluções são obrigatórias. A liderar a lista das novas tecnologias preparadas para este novo tipo de desafios, composto por um sistema de ficheiros distribuído e uma plataforma de processamento distribuída, está o Hadoop. O Hadoop é uma referência para a resolução desta nova gama de problemas, e já comprovou ser capaz de processar enormes quantidades de dados de maneira económica.

No entanto, dadas as suas características, tem alguma dificuldade em processar menores quantidades de dados e tem como desvantagem a grande latência necessária para a iniciação do processamento de dados.

Num mercado volátil, ser capaz de processar grandes quantidades de dados baseadas em dados passados não é o suficiente. Tecnologias capazes de processar dados em tempo real são igualmente necessárias para complementar as necessidades de processamento de dados anteriores.

No panorama atual, as tecnologias existentes não se demonstram à prova de tão distintas necessidades e, quando postas à prova, diferentes produtos tecnológicos necessitam ser combinados.

Resultado de um ambiente com as características descritas é o ambiente que servirá de contexto para a execução do trabalho que se segue. Tendo com base as necessidades impostas por um caso de uso pertencente a IoT, através da arquitetura Lambda, diferentes tecnologias serão combinadas com o objetivo de que no final todos os requisitos impostos possam ser ultrapassados. No final, a solução apresentada será avaliada sobre um ambiente real como forma de prova de conceito.

Key Words

Big Data, Real-time data processing, Bulk data processing, Scalable and distributed architectures, Lambda Architecture, NoSQL, Internet of things, Data Analytics.

Abstract

Since the beginning of the first decade of current millennium, it has been witnessed an exponential grow of data being produced every day. First, the increase was given to the amount of data generated by GPS devices, then, the quickly arise of social networks, and now because a new trend as emerged named Internet of Things.

This new concept, which is already a reality, was born from the premise of facilitating people's lives by having small electronic devices communicating with each other with the goal to suggest small daily decisions based on the behaviours experienced in the past.

With the goal to keep this concept alive and extended further to other applications, the data produced by the target electronic devices is however need to be process and storage. The data volume, velocity and variety are the main variables which when not over planned on the correct way, a wall is created at the point of enviabilize the leverage of the true potential of this new group of applications.

Traditional mechanisms and technologies did not follow the actual needs and with the goal to keep the concept alive the address of new technologies are now mandatory. On top of the line, leading the resolution of this new set of challenges, composed by a distributed file system and a parallel processing Framework is Hadoop. Hadoop have proven to fit under the new imposed challenges being capable of process and storage high volumes of data on a cost-effective batch-oriented way.

However, given its characteristics on other hand it presents some drawbacks when faced with small amounts of data. In order to gain leverage on market, the companies need not only to be capable of process the data, but process it in a profitable way. Real time processing technologies are needed to complement batch oriented technologies. There is no one size fits all system and with the goal to address the multiples requirements, different technologies are required to be combined.

Result of the demanding requirements imposed by the IoT concepts, is the environment which on will be relied the address of the business use case under analyses.

Based on the needs imposed by a use case belonging to IoT, through the Lambda architecture, different technologies will be combined with the goal that in the end all the imposed requirements can be accomplished and exceeded. In the end, the solution presented will be evaluated on a real environment as proof of concept.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ACRONYMS

API - Application Programming Interface

DAG - Directed Acyclic Graph

ETL - Extract Transform Load

HDFS- Hadoop Distributed File System

IoT- Internet of Things

ISO – International Organization of Standardization

JSON – Java Script Object Notation

JVM – Java Virtual Machine

M2M- Machine to Machine

NoSQL- Not Only SQL

RDBMS – Relational Database Management Systems

RDD- Resilient Distributed Dataset

SQL – Structured Query Language

URI – Uniform Resource Identifier

URL - Uniform Resource Locator

XML- Extensible Mark-up Language

YARN- Yet Another Resource Navigator

1 INTRODUCTION

This chapter presents the problems emerged owing to the big amount of data being produced everyday not only by people but also by devices. A brief introduction will be given of big data history and its underlying problems. This chapter encompasses and enforce also the process and analyse of these data needs. At the end, the work objectives will be defined and its structure presented.

1.1 PREAMBLE

According to latest studies, the amount of data being produced every day is continuously increasing. At the end of the current decade, are expected to exist up to 50 billion devices capable of exchange data between then [1]. The problem is not the amount of data produced, but the underlying needs to store, process and analyse it.

The companies have noticed that from raw data, it is possible to extract valuable information, which in turn it can be transformed into knowledge. They realized that knowledge on a volatile market can be a key differentiator factor and may be the source of its survival. New business models, changing the way enterprises operate, have started to emerge and are also already being adapted to the biggest ones [2] .

Tale advantage from raw data, it is a slow and complicate process with several stages involved. The challenge, more than how to do it, is how to do it on a profitable way. Traditional models did not follow the evolution as the needs in the same proportion. They did not prove to be capable of store and analyse such amounts of data on a so adverse and always shifting environment.

In respond to these needs, new technologies have emerged last years. From NoSQL database until Hadoop, a new Ecosystem of scalable technologies have arisen. Hadoop has been under the focus for last years, and it fits its best when faced against the process and store of data on batch-oriented way. Hadoop is the desire tool when it comes to the process and analyse of terabytes of data on a cost effective way. However, despites of being able to process large datasets, it is not capable to do that on a timely manner.

Filling outlined needs, recently started to appear the real-time processing platforms. Stream technologies, based on accumulative and probabilistic algorithms, aiming to process data while it stills on flow, presents itself as a game changer. Data thresholds identification, real-time analytical insights and quick data summarization are some of the use case application, which has led to it quick adoption.

On one side, there is high throughput provided by batch processing technologies, on other the real time metrics calculation. There is no one size fits all and on a demanding

market when both requirements are need to be leveraged, under the same solution several technologies are need to be combined.

Result of an environment with highlighted requirements is the one which will be faced under the current work. The business scenario which will be attended, is related to smart parking, which is one of the trended scenarios belonging to IoT. Encompassing both the batch and real-time processing needs, it will enquiry the enforcement of so distinct technologies to be combined as one. Based on Lambda Architecture, a solution will be expected to be built with the goal to be put into proof against a real environment scenario.

1.2 OBJECTIVES

The goal of this dissertation is the development of an End-to-End framework capable of combined past with last received results while data in flow. As a first requirement, from network, data must be gathered and mediated until the solution processing layer. On gathered results, analytical results and required metrics are required to be calculated and made accessible to external mechanisms. The information therefore will need to be represented on a dynamically built interface.

The solution under development should operate on a distributed way, and be easily extended to complex environments while adapted to different use cases.

To allow the development of described framework, the first task to be attended will be the study and familiarization of real time and big data parallel processing technologies. After overall scope analysed, as proof of concept, a solution capable of respond to the defined business case features is expected to be built with the goal at the end performance requirements can be overreached.

1.3 STRUCTURE

The current document is divided in six chapters. The current chapter presents the structure of this dissertation and its goals.

The remaining chapters cover the following aspects:

- *Chapter* 2 presents the background and state of art supporting this dissertation focussed on databases management models, parallel distributed processing technologies and data exploration;
- *Chapter 3* outlines the dissertation use cases requirements;
- *Chapter* 4 highline the proposed architecture and its implementation details;
- *Chapter* 5 draws the conclusions corresponding to proposed proof of concept;

- *Chapter* 6 evaluate the implemented work applicability and future work enhancements.

2 BACKGROUND

This chapter starts by giving an overview of the most important concepts related to data managements systems, parallel processing technologies and data analytics. At the end, business projects leveraged by these technologies will be presented and described as state of the art.

2.1 CONCEPTS

The popularization of electronic devices leveraged by its decreasing shape, price and energy consumption, while continuously increasing processing capabilities, lead small-embedded devices to be capable of process heavy data workloads on cost effective way. Empowered by network connectivity and data exchanging facilities, when combined together an endless world of unseen possibilities arise.

The exchange of information between external physical devices coming from outside environment to computing systems through network, standardized as Internet of thighs – IoT, aims to facilitate people every day life's instructing support based decision gathered by combining own and involving assets and behaviours coming from different sources. Encompassing a wide range different components and blocks in order to build the concept and make it a reality as presented on Figure 1, a wide and different set of challenges are need to be overcome.

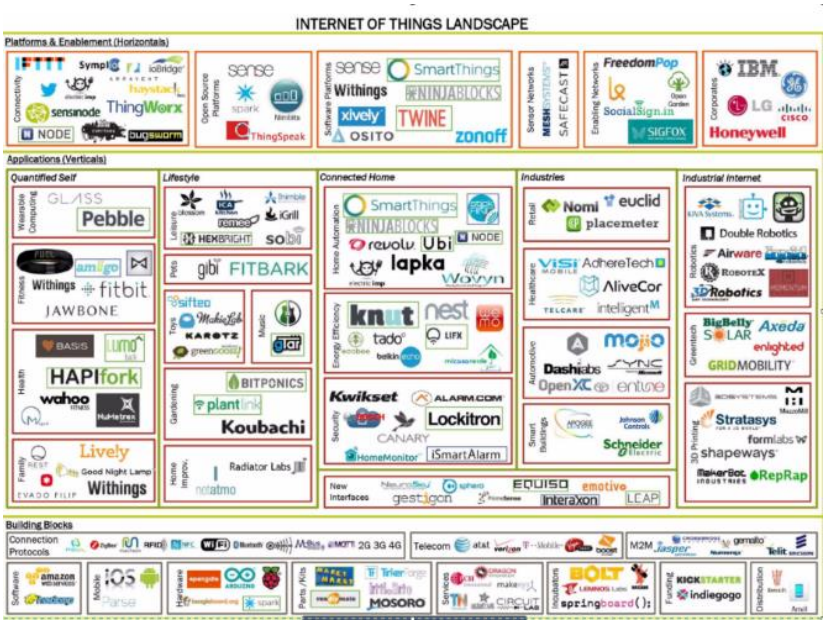

Figure 1 - Internet Of Things Landscape[3]

To allow the concept to be fully leveraged, areas ranging from telecommunication to data processing capabilities and analysis are need to be combined and overstep as one.

On a nearly future, trillions of devices are expected to be connected. Described areas and technologies does not only need to be combined and overstepped but also pushed further. As ever, boundaries are pushed over when new limits are imposed. In response to new needs, are already the noticed improvement imposed by the new network generation and data processing technologies. In common, technologies are getting simpler, lighter, more robust and scalable.

Smart cites, smart homes and smart energy consumption, are some of the main projects under development where IoT concept is applied and where a set of applications opportunities are already being leveraged.

Accompanying IoT continuously evolution, are also its underlying concerns. Security has been the main concern while this trend keeps arising. Increasing data exchange and applications monitoring, increases also the possibility of the exploitation of peoples' behaviours and private lives. Internet of things promises to revolutionize peoples live making it easier with yet unpredictable vast and numerous application, although it unweighted increasing can compromise it on a much larger scale.

2.2 RELATED WORKS

Although faced improvements on technology area have been remarkable, there still plenty of work to be accomplished. Ranged from its concept and its underlying drawbacks, there still exists a vast number of areas to be exploit and improved. Since the devices producing mechanisms and its overall management, going through the networks communication until data aggregation and analyses, a large number of areas are encompassed before data can be presented to the final user as represented on Figure 2.

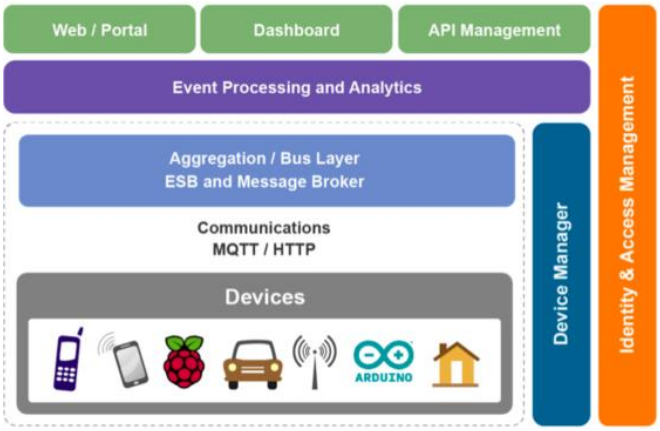

Figure 2 - Internet of things main areas hub[4]

The processing stage, one of those that has faced major improvements, is the connecting point that will connect the data prevenient from the electronical devices until its final presentation. More than a connector layer, it is the layer that will determine how data will be handled and define the success or not of the solution.

Encompassing data filtering and transforming mechanisms, data exploitation and enrichment processes are some of tasks that usually are need to be taken into account when a processing layer is thought. Motivated by the use case under analyses, the data processing stage will be the one where more attention will be relied. Based on work expected results, a solution capable of ingest data generated from external electronic devices related to Smart Parking business scenario are need to be overwhelmed.

The business model under analyses based on Smart Parking trend, it is not new and under other institution have already been leveraged by. Companies such Smart Parking[5] and Bosh Mobility Solutions[6] are example of success cases where it was correctly leveraged. Following similar thematises, under the IoT umbrella, other companies have been created. Xively[7] , Thingspeak[8] and Carriots[9] are part of the most highlighted companies. Sharing the same ideology from the companies outlined before, the last ones proposes, a standard transversal approach encompassing a wider variety of business scenarios from Smart retail to Smart Banking instead of a business focussed approach.

Deeply analysing described solution, some factors word to be highlighted were found to be essentials to all solution. Data gathering mechanisms capable of funnel and route the data becoming from different sources, it was the first requirement noticed. The second, enforced by the business specification, are the rules mappings and definition. The data visualization, was noticed to be the last aspect in common, where custom data graphics are usually made available on demand.

Leverage an on demand pre-built solution, as the enterprises described above is one of the options on the table aiming to solve the Smart Parking based work requirements. Without having the needs of going through all technology problems, a business-focussed approach would definitely be leveraged. However, despises its obvious costs, relaxing security concerns and nonetheless the delivery of the business data and its rules to an external provider are drawbacks which are not possible to be ignored. As result of this, a custom solution will require to be built from ground zero and further technological strategies are need to be analysed and correctly weighted.

2.3 Data Management Systems

Nowadays, it cannot be denied the existence of two main database management systems groups. The first, based on relational paradigm focussed on data transactional properties, has been the default choice of companies under the last years. The second,

based on different paradigms, arose with the need of distributed management systems to be built. Trading off transactional properties by more specific and specialized features, they have been complementing the traditional models ever since.

With the goal to understand the existing data management systems, both will be described on following sections.

2.3.1 Relational Database Management Systems

Formerly, all data were persisted on flat files. Despites its advantages, it was very difficult to manipulate those data and custom programs were necessary. Since 80s, when Edgar Frank Codd proposed it [10], the relational model has been considered the standard way to handle and store companies data.

In a relation model, a pre-defined schema is required, through which data proprieties and relations are defined. Characterized by a fixed structure, it offers features such integrity between relationships through constrains, and reduced redundancy through pre-defined normalization rules. The relational model, usually composed by a balanced data structure B-Tree, allows on a very effective way, operations such inserts and updates when in small volume of data. However, when under an environment in which it is necessary to handle great's amount of data, they fail to respond once they were not designed to efficiently run on distributed clusters of machines.

Over last years, only object oriented model on 90 decade, was a threat to relational model, that which were not succeed on due to the lack of agility to handle complex data and data interoperability concerns [11]. Now, not a new thread is coming but a new need. With the amounts of data being produce every day and the underline needs to process it, the traditional databases systems does not seem to be capable of handle such amount of data on an efficient way.

2.3.2 Not Only SQL

Not Only SQL or No SQL databases represents the group of databases which does not falls under the relational paradigm. The concept emerged as an alternative to relational model, where new demanding requirements exceeds the existing capabilities assured by existing mechanism.

The necessity of handle unstructured data, and horizontal scalability were the main features acclaimed by the new complex and demanding environments. Data process requirements started to reach servers capacities and with the goal to go through the noticed changes, new methodologies were mandatory to be followed.

Instead of scale vertically, using more and more powerful servers, horizontally scale across large numbers of commodity machines was required. Partitioning data through

different physical machines having each one's running one database was the pattern chosen be followed. This technique, called sharding, has been the way of dealing with demanding environments ever before NoSQL databases existence. The difference was that previous models were not sharded-aware and eventually, data inconsistency problems driven by manual maintenance errors would arise.

With the goal to make database sharding a robust mechanism, features such joins and ACID transitions would however require to be relaxed. As consequence, instead of relation models ACID properties, attached to NoSQL databases is the CAP theorem proposing the trade-off of two of the following characteristics: Consistency, availability and partition tolerance. These characteristics are not lost, but instead relaxed and replaced for optimistic control mechanisms. MVCC is an example of a relaxed properties mechanism to traditional relational databases lock replacement. From one hand, consistent problems may arise, but for other hand it is no longer needed to wait for acquire a lock anymore providing much faster data access.

There no exists any NoSQL databases providing all relation databases features or even support a widely range of different operations. It is not that it is impossible but providing all these properties to a distributed database it would make it extremely slow and automatically disabled for further purposes. Instead of it, NoSQL databases choose to be very specialized on a small set of features.

As a sequence of that, several groups of very distinct databases arose. Among the most noticeable groups are the ones based on Documents; Key-values; Columnar; Graph and Object based. When required to be chosen, the criteria should be balance based on the weight between business scenario most valuable characteristics and properties the possible to be relaxed. From a wide set of data management systems available, by group its specifications will be identified. At the end of each group, an example of the most popular management system will be provided.

2.3.2.1 Key-Value Oriented Storage System

Key Value data store are a simple hash table abstraction where given a provided key, its corresponding binary value is returned. The idea behind this type of management systems, is to provide a small set of high performant optimized operations such puts and get relaxing other not so efficient operations when leveraged by this paradigm. Due to its simple structure, they are easily scalable through horizontal sharding. Some Key-value data store allows primitive data type's storage as value. Given its low latency, they are usually used to build cache mechanisms.

Between most notable Key-value, data stores are Amazon's Dynamo, Redis and Riak.

2.3.2.1.1 Amazon Dynamo

Pioneer on this type of data stores, Amazon Dynamo origin is attributed to Amazon. Created around highly availably mechanisms, simple operations such data insertion and data retrieval given a primary key were the mandatory requirements to be met. Capable of scale out, it is schema free storing a binary blob value for every single key [12].

2.3.2.2 Wide Column Family Oriented Storage System

Column-family databases "allow you to store data with keys mapped to values and the values grouped into multiple column families, each column family being a map of data" [11].

Designed to optimize accesses to the disk, they have been used when a more flexible approach and operations are need comparing to key-value oriented storage systems, but the same performance levels are need to be kept. Storing columns instead of lines, they follow the same principals as multidimensional-sorted maps where the map is indexed by a row key, a column key and a timestamp. Supported queries features and modulation paradigm are usually more complex to deal comparing than other systems. Between some of the most notable Column family oriented data stores are HBase and Cassandra.

2.3.2.2.1 Cassandra

Developed initially by Facebook based on Amazon's Dynamo and Google's BigTable, it was release as open source in 2008, where two years later became an Apache top-level project. Based on ring architecture, diverge from common master-slave model mislaying single point of failure, providing true availability. As other NoSQL, databases is capable of scale horizontally, providing mechanisms to be easily managed across cluster shards partitioning data automatically.

Cassandra fits it's best proposes when dealing with massive amounts of write operations. Instead of perform single-unit writes, it heaps data on a memory-base structure called memTable, flushing bucket inserts when defined threshold are exceed. All the data is inserted sequentially reducing I/O increasing throughput as presented on Figure 3.

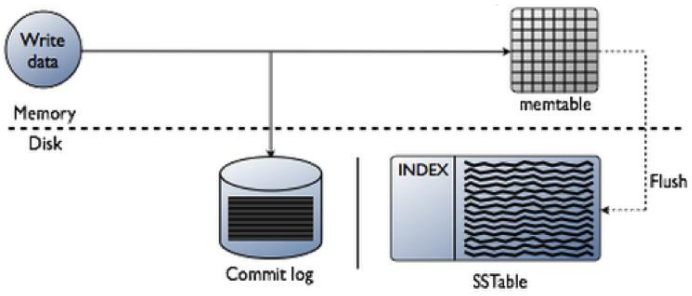

Figure 3 - Cassandra Write Mechanism [13]

Regarding to reads mechanism, with the goal to provide efficient scans, Cassandra relies on Bloom filter data structure mechanism, before reading from SSTable, discarding the file which certainly do not have the pretended data. Read mechanism architecture is present on Figure 4.

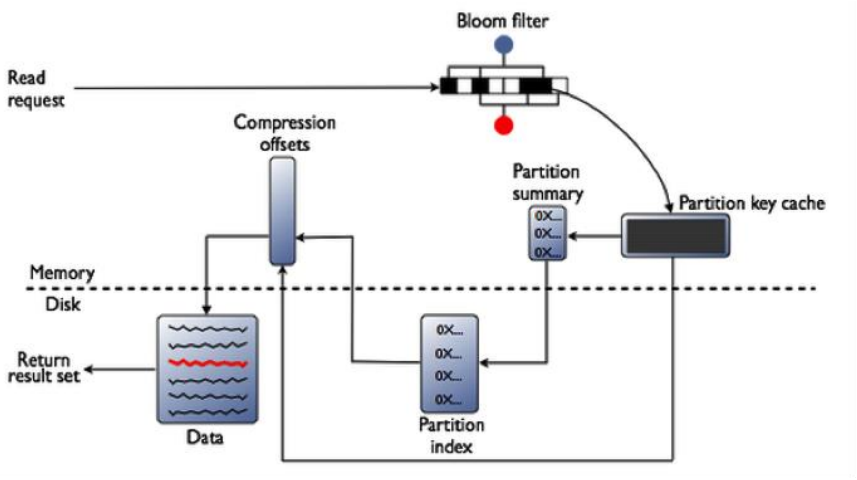

Figure 4 - Cassandra Reads Mechanism[13]

Although of provide eventual-consistency operations, it provides tunable consistency options, falling to user decide if would prefer to have all nodes answer before final reply or if one's node answer is enough to data retrieval.

2.3.2.3 Document-Oriented Storage Systems

At its core, document oriented databases leverages key-value model being optimized for cases where a document is assigned to the key. Capable of ingesting either JSON or XML documents, they offer the possibility of search the documents according its semantics like faceted search.

Usually, they are schema less, however most of them provide capabilities to validate documents according to is structure XML schema like. Most notable's document databases are MongoDB, CouchDB and BaseX.

Despite of being able of retrieve documents based on its semantic, it is not they core functionality. When most advanced retrieve features are needed such "spelling suggestion" or "more like this", Leucene based solutions such Solr or elastic search may require to be considered. There already exist databases capable of leverage these search engines capabilities such Mongo DB through Mongo Connector capable of sync data constantly updating target system [14].

2.3.2.3.1 MongoDb

Developed by 10gen, actually Mongo DB Company, released is first version on 2007. Capable of scale out based on sharding techniques, it relies on automatic failover and data redundancy features to provide high availability.

MongoDb as any other NoSQL databases has it owns trade-offs where eventual-consistence reads sacrificed in order to obtain low latency throughput.

Being a document based management system, it fits its best when dealing with documents. For MongoDb, documents are JSON like object that are data structure composed by a field and its value pair. One of its advantage, similar to other document oriented databases paradigm, is the support of most of the common native programming languages data types. This way, the data mapping between complex data structures to the database structure becomes a straightforward process.

2.3.2.4 Columnar Oriented Storage Systems

Instead of store data in rows as relational databases, columnar databases store data in columns. Taking this approach, these systems are capable of insert more data grouped by field to the same disk store page. Despites it presents some drawbacks when widely rows are need to read since more disk pages are need to be accessed, it proved itself advantageous when variant types of aggregation are necessary to be computed.

These are the models source of the well-known data warehouses. Data warehouses are the result of an expanded environment built with additional indexes, pre-aggregating data, materialized views and cubes (all combinations of aggregations calculated at a growing level of granularity where the combinations performed between all measures on fact tables and all existing dimension). As result, better analytical and write performance provided by since each disk page has more data of the same field and more pages could be written in parallel.

Characterized by its heavy in memory workload, they are capable of doing so thought bitmap binary indexes due its compression rate. As example of that is MonetDb database, which proves to beats it best when dealing with hot data - data being processed and intermediate results fit in memory.

Although, most columnar databases are different from any other categories described so far. Despite it does not seems to fit on most NoSQL databases characteristics, they are an important component of a modern heterogeneous environment. Columnar-based solutions are emerging lately on market and druid and Google’s Dremel are some of the most notable examples among.

2.3.2.4.1 Druid

Hopeless for a distributed real-time analytics system, when no solution seemed to fits required needs, Metamarket's decided to build their own.

Druid is composed by three main different components: Real-time, Historical and Coordinator nodes. When data first arrive through its ingestion system either by real time nodes or Tranquillity API, it is buffered in memory until stored on Historical Nodes. Historical nodes are system workhorse and the place where data it is handled, metrics calculated, bitmap indexes made, data pre-aggregated and compressed to a columnar oriented format. The output results of historical nodes are call segments - druid fundamental storage data structure unit. Segments will be further stored on configurable deep storage. Alternately data can also be ingested on batch oriented way, which when configured it is capable of replace existing segments for the pre-defined time. Released the first version under current year, some major companies such Netflix are already leveraging it. Overall Druid picture is presented on Figure 5.

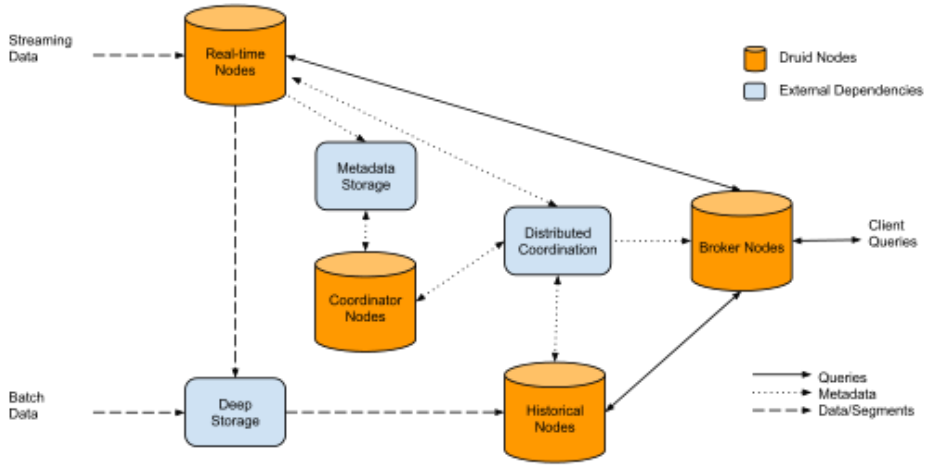

Figure 5 - Druid's Architecture[15]

2.4 PARALLEL DISTRIBUTED PROCESSING SYSTEMS

Despite NoSQL being able of handle and process big amounts of data, parallel distributed processing systems ambition to process and analyse much more. Not concerned with consistency or transitional properties as first priority, its focuses is to provide intense workloads of massive data processing on distributed cluster of machines.

When talking about parallel distributed processing systems, nowadays Hadoop Ecosystem is an inevitable asset. However, it is not the first trying to address parallel processing challenges. Among the most important systems, stand out grid computing and

high performance computing systems. The most important differences between previous systems with Hadoop will be stat when described next.

2.4.1 Hadoop

Hadoop stands as an Open Source project written in java built to address problems such scalability, availability, and data integrity.

Levering parallel computation paradigm, it is able to process massive datasets with hundreds of terabytes or even petabytes of data across clusters of commodity hardware. Hadoop encompasses two principal components: Hadoop Distributed File System and Hadoop Map Reduce. When the two components combined, due to its distributed file system, it guarantees scalability and auto healing, and with its MapReduce framework, it allows the process and analyse of big amounts of data in parallel manner. It also allows the data to be processed on the machine where it resides called as data locality, instead of have to move all the data from network like previous existing system.

More than a project, Hadoop is already an ecosystem of projects. Its commonly refer Hadoop as any of those projects built around of its base. Some of the Hadoop ecosystem projects are present below on Figure 6.

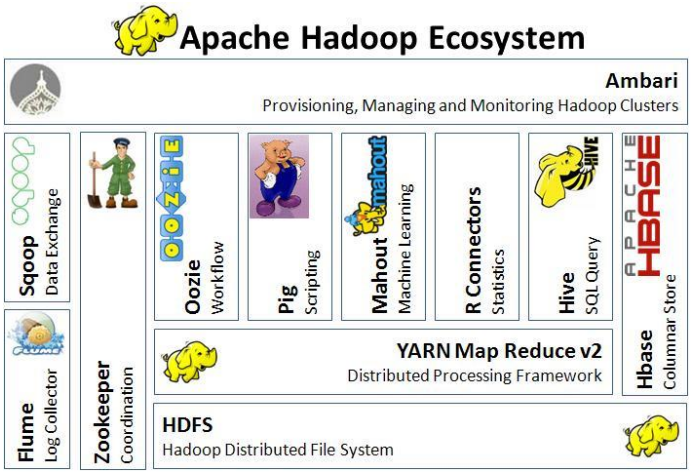

Figure 6 - Hadoop Ecosystem [16]

2.4.1.1 Hadoop History

Hadoop is a branch from an initially project called Nutch. Nutch was, originally, created by Doug Cutting in 2002; also the creator of Apache Leucene. When first born, was meant to by a web crawler. During its creation, several problems raised, which were eventually resolved later based on Google papers. The project successfully achieved their goals and kept growing until 2006, when it was offered a dedicated team at Yahoo! The transition was the moment when Hadoop first born emerged from Nutch.

Two years later, there was created an open source version under the terms of Apache foundation. Nowadays, not only Yahoo! leverages Hadoop, but a set of other companies, like Facebook and Amazon.

2.4.1.2 Hadoop Distributed File System

Based on Google paper [17], Hadoop Distributed file System - HDFS, is a file system distributed across several machines. It was built to store and transfer big amounts of data on commodity hardware with fault tolerance.

Hadoop distributed file system can not be compared or qualified as a database since it does not allow some basic CRUD operation or provide transactional processing. Rather than seek pattern characterized by databases, it was built around transfer pattern paradigm where latency is relaxed in order to big amounts of data, can be ingested to be processed.

HDFS, is similar to regular file systems, such Unix systems. Moreover HDFS can be easily leveraged being mounted on a regular local file system directory thought FUSE (File System in User Space) on a NFS gateway [18]. However, it present some significant differences [19], among them the blocks size - minimal amount that can be read or write, which can be extended up to 1GB, and the way file system metadata is handled, storing it on dedicated machine instead of on local file system.

The block size presents itself as one of the most import concepts of HDFS as it will be the abstraction layer to the distributed File System (more easy to replicate blocks rather than files), and it will responsible to define the input size and number of Map Tasks when leveraging Map Reduce Paradigm. Being on a standard system the blocks size capacity around 512Bytes, HDFS default size is set to 64 Megabytes, extendable to 1 Gigabyte. Blocks' size choice can be made based on two main aspects. The first is the processing time (smaller block sizes are computed more efficiently as there are more blocks to run on parallel). The second it is the task's scheduling time (where block sizes are inversely proportional to task management time, which smaller blocks sizes would increase the overhead to schedule such process overtaking the first aspect stated, the processing time). The default HDFS blocks size, is optimized to most use cases however is should be evaluated and customized if needed.

2.4.1.2.1 HDFS Architecture

Based on Master slave architecture, HDFS is composed by two main components, which are NameNode and DataNodes as presented on Figure 7.

An HDFS cluster contains a single NameNode, on which is relied all the file system metadata. Remembering the block concept, as the file system abstraction layer, instead of

files, the NameNode is responsible for manage all the file system blocks-files mapping in its memory. File system tree structure is persisted on two files: EditLog, where all occurred changes on filesystem are persisted, and FSImage, where the entire file system blocks-files mapping kept.

Unlike NameNode, HDFS's cluster has several DataNodes, usually on for each node. DataNodes, are the system workhorses. They are responsible for store and retrieve the information when ordered by the NameNode. Additionally, they are also in charge of periodically reporting its condition (list of all blocks on DataNode) to NameNode through asynchrony heartbreak's messages.

Some of the most important characteristics, where HDFS was built on, are data reliability and availability. In order to address these properties, HDFS provide block replication. The replication factor HDFS's default value is three. Usually, being replica factor three, the data is spread to two different racks, being the first responsible for store one and the other remaining two on different nodes.

Leveraging this concept not only reliability and availability properties are guaranteed but also resources optimization. Resource optimization can be achieved by HDFS when advantaging the rack-aware policy. Rack-aware policy allows system to be aware where data resides. This permits that most closed replicas are request, over the remains.

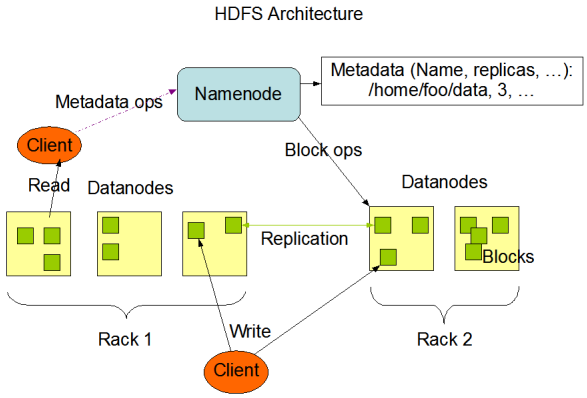

Figure 7 - HDFS Architectue[19]

Being all cluster metadata information relied on a single node, the NameNode, question such as file system availability and scalability are being compromise. This question has been addressed in versions prior to Hadoop 2.0. Regarding availability questions, once NameNode fails or need to be stop for maintaining reason, the cluster would be unavailable until NameNode restart over. After Hadoop 2.0, this question is addressed running two redundant NameNodes in the same cluster, being one of them in stand-by allowing fast failure recovery. Regarding to scalability concerns, as all file system mapping it is on memory, this will end with a failure scenario. To resolve that, in Hadoop 2.0 multiple

NameNodes are allowed, having each one their block pool managed independently from other block pools- HDFS Federation as can be observed on Figure 8.

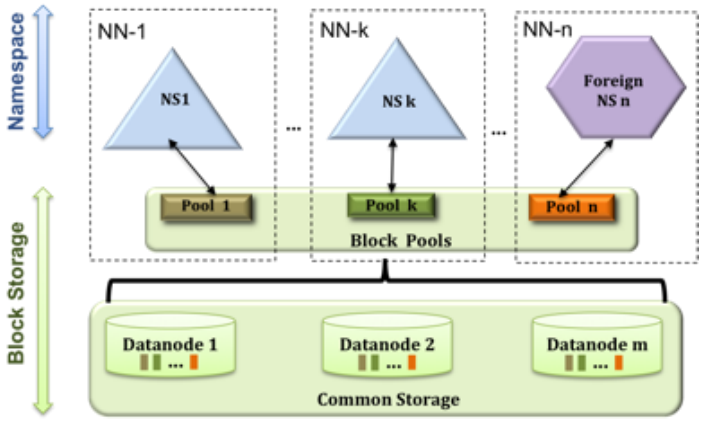

Figure 8 - HDFS Federation[20]

2.4.1.3 Hadoop Map Reduce

Based on a concept originally founded on functional programming languages, Map Reduce is a programming model designed to process large datasets on a distributed-way. First introduced by Google [21], it splits the problem into small pieces and send each one to different machines enabling the problem to be processed in parallel. Later combines all the results delivering the final solution.

Map Reduce paradigm allows the user to focus on data processing and analyses. More than a technology or a feature it is a constraint that enable all kinds of applications to be processed across the cluster without having to deal with concurrency, robustness or scalability problems [22]. Capable of support all types of algorithms, it allows all types of applications to be written on.

Map Reduce Paradigm can be divided in two stages: Map and Reduce. In order to take advantage of the paradigms both Map and Reduce operations functions need to be written and engaged in form of acyclic graph - DAG. Representative mechanism example can be analysed on Figure 9Figure 9 - Hadoop Map Reduce .

Each of these phases is further divided on sub-phases. Map phase is constituted by four sub stages: reader, mapper, combiner, partitioner and Reduce composed by two: Shuffle, reduce, and output.

The process starts by reading data for a given source. Taking into account the volume of data, it is divided into chunks in order to each one being distributed on a parallel-way by Map operations. When dealing with HDFS as source, data locality is leveraged, being data processed by the server where it resides. On mapper phase, all data are set in form of key-value. Mapper phase is responsible usually to either load, parse, transform or filter data.

Combiner follows mapper action, and it is an optional phase, only possible on specific cases taking action after mapper and acting as a locally reducer. When harnessed it saves network amount of traffic being exchange across the cluster. The Map operation ends with partitioner. Partition stage is responsible of the results store on local file system partitioned by its key, that way avoiding data replication opposite to HDFS persistence.

Accomplished Map operation, shuffle is the first Reducer operation. It starts by downloading all the intermitted results from partitioner phase. Shuffle's goal is organizing all data in order to send tuples with the same key to the same reducer. Reducer phase is responsible for combine inter-mediate results with the goal to compute final results. The process ends with data being written to HDFS files.

On most simple scenario, there is one map and reduce operation. Nevertheless, faced with higher complex problems, other approaches must be taken into account. Instead of write a widely complex Map Reduce, operation write multiple simple Map Reduce tasks would be a better approach. The solution is to write an acyclic chain of Map Reduce operations whereas the output of one stage is the input of other. Map Reduce chains can also be inverted - chain folding, in order to write more complex and improved solution efficiency. As result, could happen to exist chains such Map-Map-Reduce or Map-Reduce-Reduce among others.

Develop and manage a simple Map Reduce application could be a very expensive and long process. Allied to a constraint paradigm it is a low-level API, turning results later to appear. Without have to mention how arduous would it be the development of a complex application, a wall is reached under such requirements and other options must be taken into consideration.

Addressing this problems, arose projects such Hive [23] and Pig [18]. These projects leverages Map Reduce on their internals making them own workflows given a higher level abstraction language. Providing fully optimized complex workflows, rarely do not prove to be the best choice when compared with Hadoop Map Reduce low level API. Even really helpful for most common operations and applications, still exists a range of unsupported operations left. On these cases, when most flexible and specialized applications are need to be built, projects such like Crunch, Cascading and Spark should be taken aboard.

Lower level than projects such Pig or Hive, but higher level enough over Hadoop Map Reduce API, they provide the desire flexibility and maintainability given a reasonable learning curve and development time.

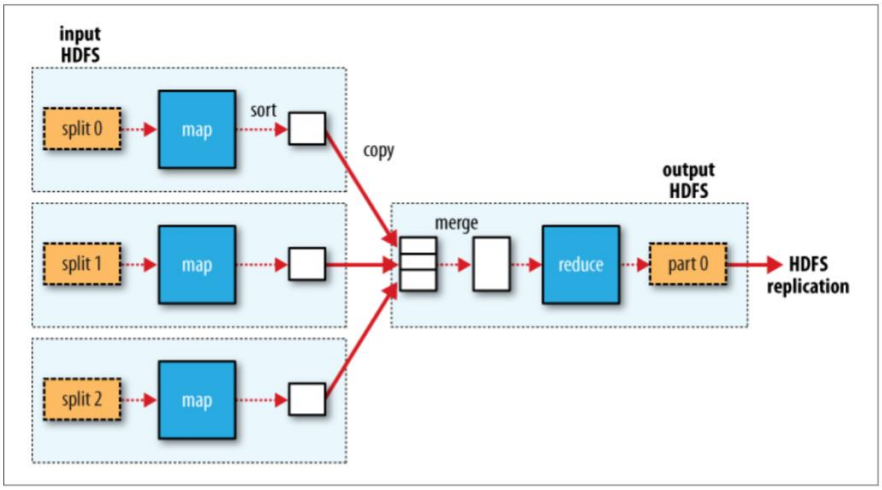

Figure 9 - Hadoop Map Reduce [18]

2.4.1.3.1 Architecture

Being the process workload distributed across a widely distributed cluster of machines, with the underlying needs of coordinate, assign and synchronize all Map and Reduce stages, a component must exist to handle these questions. Prior to Hadoop 2.0, a Map Reduce job, on a single component was relied all described tasks.

Job tracker was responsible for scheduling and distribute Map Reduce tasks for later, Task Tracker, another component localized on each cluster node, process those tasks as presented on Figure 10. Existing a single Job tracker per job, this would address either availability problems owing to the fact of exist a single point of failure, either scalability restrictions.

Besides all these limitations, resource manager would be only capable of manage Map Reduce Application left back other paradigms taking advantage of clustering infrastructure.

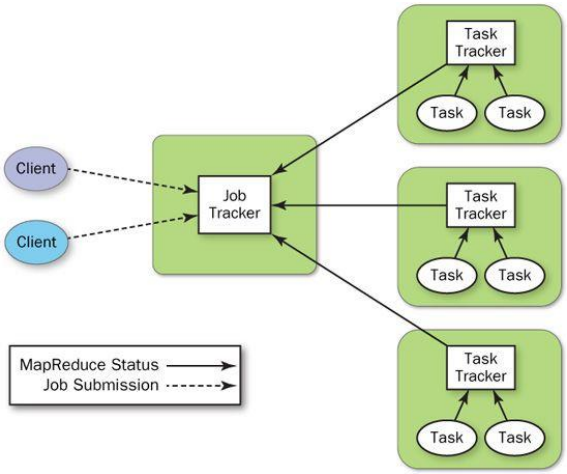

Figure 10 - Map Reduce Architecture[24]

2.4.1.4 Yarn

Yet-another-resource-navigator or Map Reduce 2, name given due to most of the code built on was reused from Map Reduce, it can be described as “cluster operating system that provides the essential services for applications to take advantage of a large dynamic and parallel infrastructure” [24].

Evaluating Map Reduce management process, Yarn presents some major improvements concerning to cluster management and support. Addressing described problems on last section Yarn splits previous Job Tracker responsibilities ((1) resource management and (2) scheduling/monitoring) and reassign them into two new components: Resource Manager and Application Master.

With the appearance of this new component, resources are allowed to be negotiated between Application Master and Resource Manager, enabling specific requested resources container to be booked to the desired task operation on Node Manager. Having a Resource Manager monitoring the information about percentage of use of each node in the cluster, it enables the tasks scheduling and distribution to happen more efficiently. Resource Manager, given the cluster properties, chooses the nodes, which most well fit the desired properties for a given job. On Node Manager, a container will be booked, with the specific memory and processing capacity required.

Decoupling cluster resources management and scheduling capabilities will enable Hadoop to support new range of applications based on paradigms other than Map Reduce as presented on Figure 11.

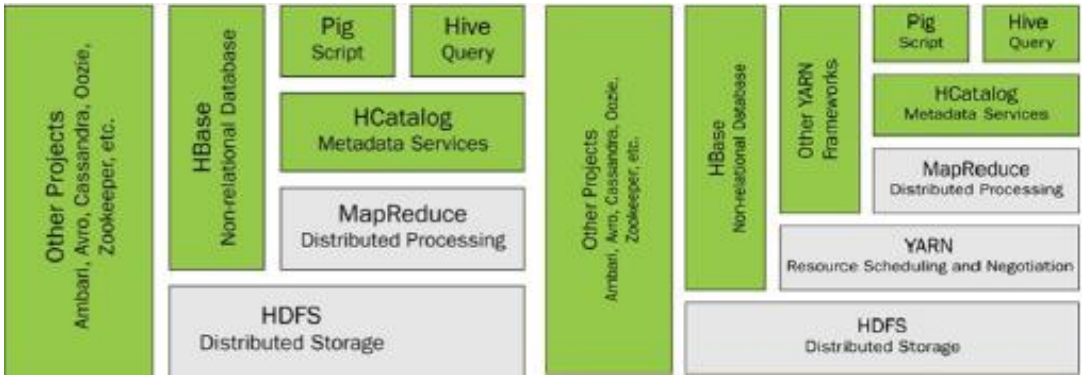

Figure 11 - Differences between prior Hadoop Version and Hadoop 2.0 using YARN[24]

Remembering Job Tracker, it was presented as a single point of failure. In order to High availability concerns, YARN even presents a single Resource Manager, it support a second Resource Manager on stand-by allowing for recovery on cases of failures. Overall YARN architecture can be analysed on Figure 12.

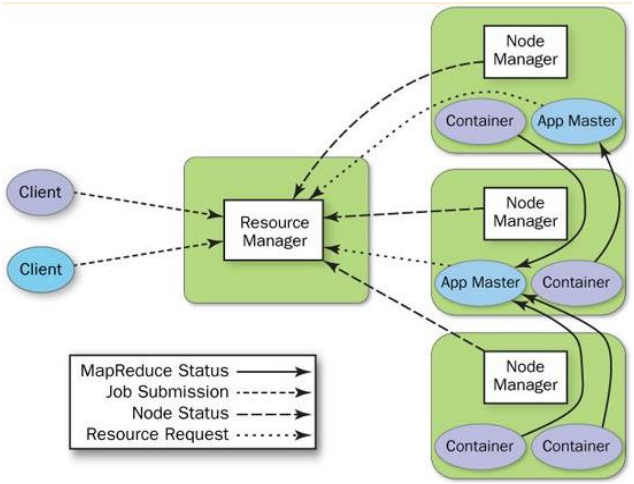

Figure 12 - YARN Architecture

2.4.1.5 Spark

Hadoop has been the leader solution of big data market last years. However, it has well known liabilities. Being a demanding increasing market, other technologies had raised to reclaim its share of the market. Hadoop gained interest of several companies, and a lot of products have been built around either to complement it, or to replace some of its components.

Spark is a fast data processing engine that complements Apache Hadoop Ecosystem, and aims to replace one of Hadoop components - Map Reduce. Since ever, specialized systems have been built around mixing and pipelining technologies in order to best fit it is proposes. The problem with specialized systems comes with the price of third party integration mechanisms. Apache Spark aims to overpass these constraints.

Combining batch, stream and analytics on a unified framework leveraged by a higher level API, it replaces the previous needs of mix several different technologies as presented on Figure 13 by improved applications productivity given Spark own ecosystem represented on Figure 14.

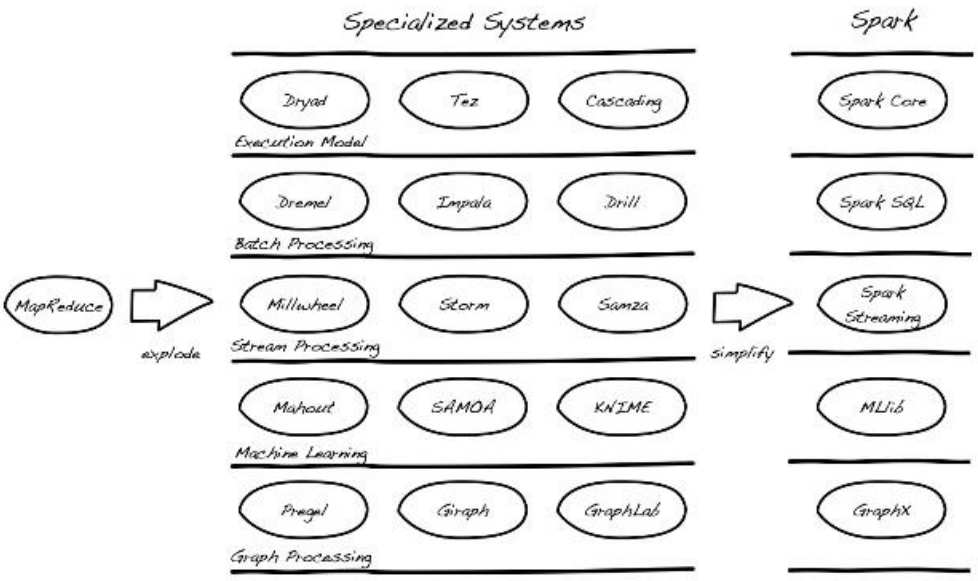

Figure 13 - Spark flexibility[25]

Based on 2007 Microsoft paper [26], Spark has graduated on 2014 as an Apache top level project. Leverage memory ever possible, it improves some previous Hadoop Map Reduce intermediate steps. Instead of having to write and read from disk between each Map and Reduce stage, it takes advantage of memory when data fits in, improving that way drastically the process efficiency. Besides, taking advantage of cache mechanism, interactive algorithms can be improved significantly too. Benchmark test demonstrate that Spark can bet Hadoop Map Reduce jobs over 100 times when in memory[27].

Spark main abstraction is RDD - Resilient distributed dataset, which is fault-tolerant "collection of elements partitioned across the nodes that can be operated on in parallel"[28]. Being the collection of elements immutable and transformation history, lineage recorded, any step can be re-computed and re-reschedule in a fault-tolerant way. Similarly to Hadoop, data locality and rack awareness are leveraged on Spark too. On RDD

can be applied two types of operations, which are transformations and actions. The transformations are functions such map or filter, and actions are functions such reduce.

Given the high-level provided API, unlike Hadoop Map Reduce, complex DAG pipelines can be written easily. Below are a sample of application written on left with Hadoop Map Reduce and right with Spark API.

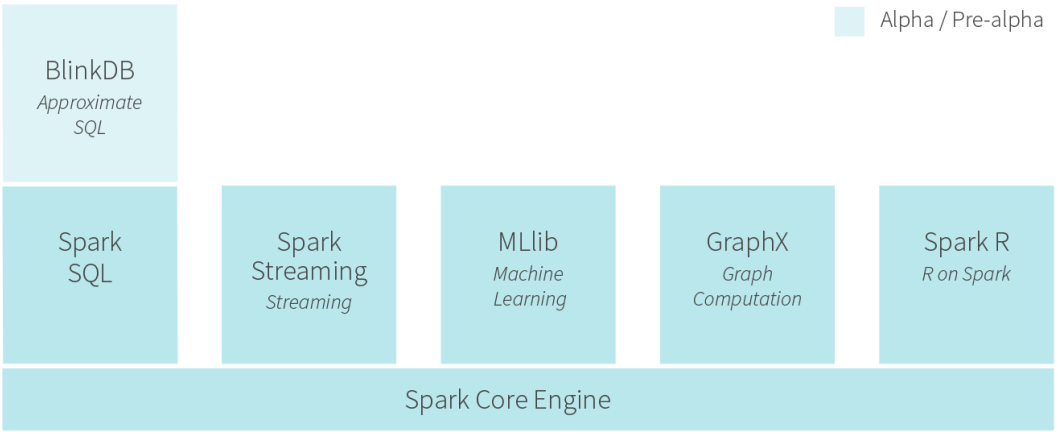

Figure 14 - Spark Ecosystem)[29]

2.4.2 Data Analytics

Massive amount of data produced every day prevenient from everywhere. However, it is just raw data and raw data is all it is. Even if it presents itself as useless, when we try to look closer, we start to realize that it is not as useless as we thought on the first sight. Every small record of data has the potentiality of being something more, which when brought together with other records, sometimes it becomes indeed something more. This something more, is called an insight and when leveraged on the right way, it could be the starting point of an unseen opportunity missed for others who did not look further for that useless piece of data.

A missed opportunity is all it takes for losing competitive advantage, which in turn is what sinks an uncompetitive business. Grounded on information Era, where everyone has access to everything, more than ever it is mandatory to look further in order to gain insight so it is possible to earn leverage to others on market.

Discover insights from raw data it is not a simple or a straightforward process. Most of the times, data is dispersed from different sources on different formats, and thus imply some pre-processes to be accomplished first as described on Figure 15. Those processes, many times referred as ETL (Extract, Transform and Load), are mandatory to standardize data on a single place, with the goal of information could be extracted easily on a cost-effective and efficient way. Once these steps overtaken and all data placed on a single repository, generally on a Data Warehouse, data is ready to be explored.

Data exploration process, usually called statistical analyses is the next step need to be taken. Usually, it is present and divided into two main categories: Data Summarization through descriptive statistical analyses, and Decision support aiming through inferential analyses.

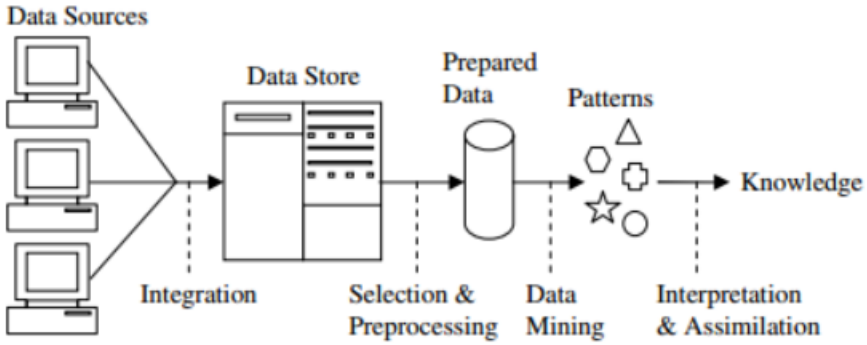

Figure 15 - Data flow chain[30]

2.4.2.1 Descriptive statistics

The descriptive statistics consist on the use of simple methods and mechanisms such average and standard deviation aiming to resume dataset on an objective way. With Descriptive statistical analyses it is pretended obtained a clear perspective of the data it is being worked on. Most usual way to overpass this step is to benefit from already existing mechanisms within Data Warehouse systems leveraging the already outlined advantages on *Chapter 2.3.2.4 - Columnar Oriented Storage Systems* .

Followed by these mechanisms are often the third party dashboards. When used together, a clear understanding of data is enforce assisting on an accurate way the next steps needed to be taken on the exploration flow.

2.4.2.2 Inferential statistics

While descriptive statistics aims first describe the data it is being work on, the inferential statistics aims to explain it. The inferential statistics, pretends to infer results given the noticed random variations with methods such hypotheses test, correlations analyses and interpolation methods. Having in mind the results obtained on previous described descriptive statistics, those results can be extended to understand and predict unobserved values with decision support solution.

There exists a wide range of fields of study in charge of decision support solutions, between the most notables are machine learning and data mining. Both based on statistic models, the difference between them are that the last is focus on practical deployment aspects of the first.

Data mining models can be divided into two main categories: Supervised and unsupervised learning. The difference resides on the way the models are build. On Supervised Learning, data models are built around a set of expected results – labels, which are provided by the training dataset. On Unsupervised, oppose to supervised learning training dataset does not contain the expected group result or categorization.

The supervised learning can further be segmented and applied into diverse applications. Based on the expected result, discrete or continuous, classification or regression models can be built based on.

Having in mind the big data problems, bellow will be described the most notable frameworks built around big data ecosystems, in order they can be leveraged to assists described goals on *Chapter 3.1 - Use case Description*.

2.5 REAL TIME PARALLEL PROCESSING SYSTEMS

So far, the only concern has been processing large amounts of data on batch-oriented way aiming to make data reports, or finding patterns to understand the data under study and help support decision making. Yet, still there are cases where it is necessary to obtain information in real time. Therefore, arises the need of processing data while it is on moving, i.e. streaming, to gain immediate insights. Data thresholds identification, real-time analytical insights and quick data summarization are examples of the use of case application requirements.

Recalling the CAP theorem, where from a set of defined characteristics some would need to be compromising in order to best take advantage of other, depending the resources were most valuable on use case, on Stream processing there are also some trade-offs needed to be applied. On stream processing, the most import asset is the latency. The is data required to process as it arrives in order to avoid congestion and to provide useful insights. However, to be able to process all data and acquire insights as it arrives it is a challenge for any company. It is not possible to extract every piece of information in real time and cross with all previously received data or cross all received data within a complex rule engine.

Processing streaming data is however a challenge and on existing systems, given its earlier arise, trade-offs are need to be made. Accuracy is many times, the first property to be lost. Filtering the amount of data to be processed can be the first step to reduce latency. However, as mentioned, accuracy will be lost and important data can be discarded and never reach out to the system.

There are therefore, other concerns that Streaming process imposes and there are some concerns to be overcome. batch algorithms applied on the batch analysis will not fit on this approach and as consequence are required to rethought. Instead of a batch oriented

analysis, a new and mandatory approach it is need to be called into action and previously implemented algorithms shifted to update incremental algorithms.

2.5.1 Probabilistic Algorithms

When faced with the simplest problems - counting problems, on a non-streaming way, an approach relied on elements persistence together with a counting process would be all it takes to the problem solving.

On a streaming way, while this is a problem that naturally could be resolved replacing persistence mechanism by an incremental counter, when the numbers of distinct events are required to be counted increases, on a greater scale, memory problems can arise.

Having ten million of events on a dataset, being one million of then distinct would require the spent of seven Mb of memory to store all counters on collection to make possible to answer the problem. Leveraging probabilistic structure trading off four percent of error, would only be require 5 Kb of memory to answer the stated question [31].

Trade off space and performance for accuracy is the geneses of the probabilistic data structures. First introduced by Munro and Patterson, later formalized and popularized by [32], it has been an active area of research ever since. Common problems transversal to most of stream processing research community are the heavy hitters' problem, the counting distinct event and the partnership belonging. There exist also probabilistic structures for calculating sums, averages min and max [33].

The Count-Min Sketch[34], is an example of a probabilistic algorithm. Created by Graham Cormode and S. Muthu Muthukrishnan, Hash tables are used to count the events frequency on a sub-linear space. The idea behind the data structure is, providing a set of input parameters based on the amount of space willing to be wasted, the count value is stored on several Hash Maps, which at the end through simple retrieve calculations, expected values can be guaranteed by the minimum existent value between the several created Hash Tables.

The Hyper Log Log[35] is an example of other popular probabilistic data structure. Rather than frequencies count, its goals are to calculate the number of distinct values. At its base, similar to Count-min Sketch, it relies on multiple Hash Tables to decrease of the space needed to obtain the desired result. The calculation in the end is made based on several Hash Tables group sets with similar distribution.

2.6 DATA BROKERS

A mechanism is need to transport and deliver the data prevenient by external source producer mechanisms until the processing layer. Recalling, it has already reached a time when billions of mechanisms are producing and flooding data to everywhere.

Being each one producing data under difference protocol, a funnel mechanism is required to combine all the different sources with the goal to diminish further adaptation processes to further solution elements and distribute it under an asynchronous and fairly way. Queuing mechanism are the ones who best feature to the described needs. Built around a set of reliable requirements, they are the perfect intermediary point. Tested into proof under distributed environments, shield by fault tolerant mechanisms, are highlighted tools, such Kafka, RabbitMq and Flume.

One of the streaming processing challenge is to ensure that data arriving from different sources turns available for its processing fairly while it stills on movement. Recalling that processing engine could fail, since the data is being recovered from network and delivered to processing, all that data on flow will be lost. Ensuring data exchange thought acknowledgment likes protocols is a common solution proposed under throughput trade-offs.

Leveraging all described properties, Kafka is an example of such technologies. Capable of funnelling data from different source and make it distributed by different brokers across a distributed cluster data will be partitioned based on its topic on a round robin scheme or any other pre-defined partition function making it available later to be consumed on a parallel way. Built around a provide a staging system for all data between its intake and deliverable, ensuring which data could be reprocessed when processing engine fails, it demonstrates to be a technology to be further taken into consideration when similar requirements are met.

2.7 DATA VISUALIZATION

The data presentation mechanisms, even not being the focus of the current work, are always an element word to be highlighted. The most important information when not leveraged on the right way, improved by the data visualizations principal's heuristics will not ever be fully used and its potential will be wasted and lost.

Drawing the principal gathered results, small graphical representation is expected to be constructed. For custom data representations a big set of libraries are available on demand. Between the most highlighted are outlined tools such D3[36] and Highcharts[37].

Both built around a set of pre-built charts, they should be chosen by the customization require level. D3 is an example of a completely customizable tool, but by other hand, it

requires more set-up time. Highcharts by other way, customizable enough do not require the same start-up needed time. A trade-off factor should ever be weight before any choice can done.

3 USE CASE EXPLORATION

This chapter starts by giving an overview of the use case requirements. Based on outlined use cases, data context will be explored and enhanced until full data comprehension.

3.1 USE CASE DESCRIPTION

Cars Parks are continually updating it status either a car enters, either a car leaves the park. On effort to provide useful information to drivers or stakeholders, both must be aware of current situation with last updated information. Instead of having a parking monitor periodically updating park occupation scenario, through programmable devices same information can be extracted by a set of defined sensors that in turn through network, gathered information will be published on.

Similar to described use cases on *Chapter 2.2 - Related works*, cars Parking monitoring and sensing is one of the use case which falls in the new M2M area trend which when leveraged on a correct way, useful and even strategic information can be extracted.

Park holders, and stakeholders, more than simple information gathered are interested on analytical reports and data summarization in order to identify useful patterns. Drivers are concern on time and fuel lost to park their cars. To allow published data to be lead to useful information, where insights can be extracted, it needs to be first transformed, enriched, analysed and processed. The present use case, which will be used to build the proof of the concept, is based on real data prevenient from Aveiro University parks.

Above are the proposed main requirements:

- Calculate analytics with different types of granularity
- Enrich and correlate data to enforce data context on analytics
- Patterns identification aiming assist decision making
- Real time thresholds identification based on past data
- Real time monitoring and alerting
- Top K parks with higher flow over time and day
- Active parks identification

It is expected to be built an end-to-end framework capable of responding to outlined these requirements. The framework must be built since data arrives through broker, encompassing ETL, data processing and analytics phases until the information be presented to final user.

3.1.1 Data Source

The involved data is derived from UA cars park monitoring system, and it is concerning to park's occupation by time. The data, related to 10 different parks stations, is arriving continuously on an unreliable bus system affordable under authentication service.

For studies propose a dataset was formed through a first written application *Chapter 4.2.1 - Collector Data flow* capable of listen the desired data and storing it on a distributed File System. It encompasses data since 15th March, the time of the document closure, 1st September 2015, however, due to bus system fault-tolerant semantics and reliability there exists periods of times were data were lost given system failure.

3.1.2 Data Structure

The data received from Aveiro University parks monitoring system arrives on JSON format. Each tuple received is composed by a topic that identifies unequivocally the park system who originated it, and a payload containing information such the time when event was triggered, the park occupation and the park geometric coordinates. A complete data tuple is below on Figure 16.


```
{       topic": "sMfJISMyfWHgWRIaeCJWtO/8xiwQclsQD5D2yXRmBjJLk/rjVXZESwV0Vw1vkdUz5RBC/xi1Ras0YjmUUmjUOltQdmA/P15",
"payload": {"capacity": 188, "name": "ESTGA", "timestamp": 1427717431, "free": 164, "longitude": -8.443438, "latitude": 40.574671, "occupied": 24, "id": "P15"},
"device_uuid": "xi1Ras0YjmUUmjUOltQdmA", "timestamp": 1427717432, "group_uuid": "rjVXZESwV0Vw1vkdUz5RBC", "tenant_uuid": "sMfJISMyfWHgWRIaeCJWtO",
        "tags": ["parking", "spaces", "capacity"], "product_uuid": "8xiwQclsQD5D2yXRmBjJLk"
}
```


Figure 16 - Data structure

3.1.3 Data exploration

Before trying to solve the described use case right away, it is important to understand the owed data first. As stated on *Chapter 2.4.2 - Data Analytics*, data analyses is vital and one of the most important process to any organization, where from it, information can be extracted leading to knowledge.

Data exploration process will be done through the two given main categories already described: descriptive statistical analyses, and inferential analyses respectively. The first will be used to obtain a clear perspective of the data it is being worked on and the last, to infer results given the noticed random variations whose results will be extended to predict unobserved values with Data Mining methods.

3.1.3.1 Data Correlations

In order to understand the data flow, the first thing to be done will be the search for the fields who better explains data variations. Being data based on time, a strong assumption can be done which time has an important significance power on overall dataset

results. Time series analyses were made on an attempt to find either tendency period or a cycle. After data being summarized and average values calculated by different granularities, those who better correspond to meaningful results are presented below. The graph presented on Figure 17 representing a first business scenario overview, shows the average occupation aggregated by hour along a sample of random selected days on a given random selected park.

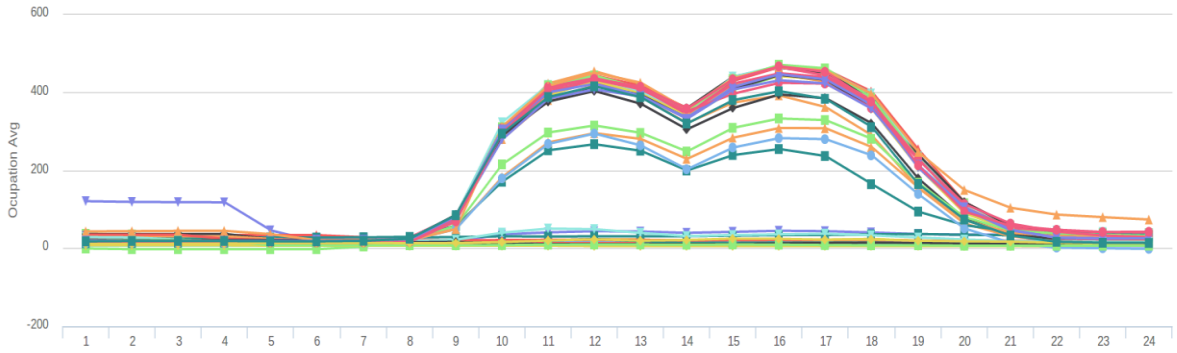

Figure 17 - TimeSeries Agregatted Data

From a population of parks, the samples under analyses now and further will be extracted given the following assumptions:

- Parks under analyses are strongly, and positively correlated, given Spearman's coefficient correlation (chosen instead of Pearson coefficient correlations because of the results nonlinear behaviour).
- The population is uniformly distributed and samples were collect following Poisson distribution.

Analysing the graph, it can be inferred which the assumption stating which time has a significate descriptive power on results cannot be dropped, and so the average of persons per hour for a given day can be classified as field of interest to help explain some of the variations, and a field to be taken on account on further analyses.

A deeper analyse on Figure 18, shows that there are three distinct groups of variation along the day. With the goal to explain the differences between groups, a deeper analyse based on more fields were done.

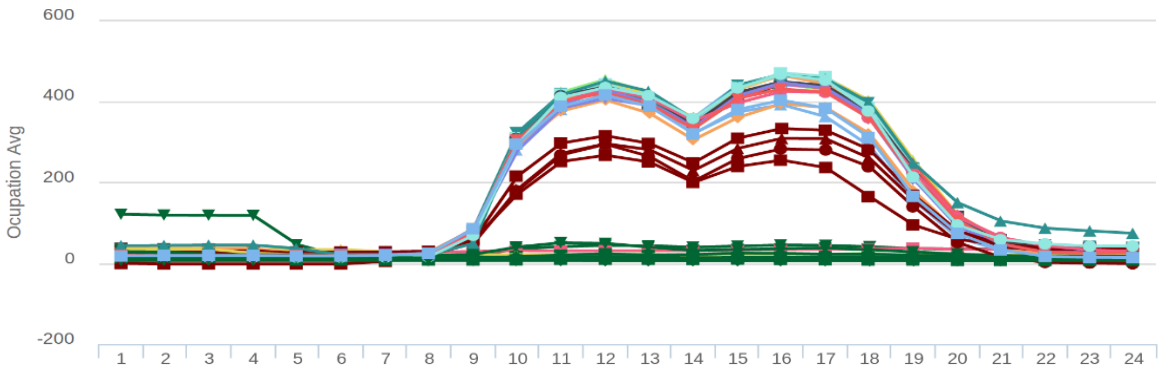

Figure 18 - TimeSeries Agregatted Data Segmentation

New fields were searched with the goal to understand the observed variations between the different groups, and the one´s with more discriminant power were grouped by noticed colours: green, brow, and the remaining colours.

The green labelled data are concerned to results occurred during weekends, brown labelled data to holydays, and the rest of colours correspond to non-holydays and non-weekend days. The present graphic brings new assumptions:

- Weekend days are different from weekdays
- Non weekdays are different from holydays

Regarding to the first assumption, which states that average number of cars by hour on weekends is different from the average on weekdays and holydays, appears to be significant and so cannot be discard at this point. The last assumption which states that average number of people by hour on non-weekdays is different from the average number of people on holydays it is not so significant and in order to make any decision regarding it, further analyses need to be made so that cases such coincidences could be diminished. Test hypothesis were conducted between two factors vectors. With the goal to accept or not the second statement, Pearson's chi-squared for goodness of fit was calculated. The obtained result concluded that this option cannot be rejected and non-weekdays can be significantly different from holydays.

Given obtained and described results, the following assumption cannot be rejected and can be stated:

1. Hour of day has a significate descriptive power

2. Weekend days are different from weekdays
3. Non weekdays are different from holydays

Described analyse were the first step to analyses the data under subject. With the goal to look for new insights, further tests would need further to be conduct looking for new independent variables trying to explain the further unseen patterns.

After dataset analysed, data was enriched, and crossed, with external data sources. It was already done before but without significant expression when analysing the discriminant coefficient for holydays vs Non week days, and it was done again for weather, following the assumption that on rainy days, the number of cars would be significantly different from sunny days. On both cases, data were enriched during the online gathering stage crossing online data with public web services API.

One last assumption promised to explain the number of cars entering and exiting of parks during day. The results obtained following last described assumption, were not so different graphically, and a new hypothesis test will have to be conducted. New hypothesis test was done, and the results, once brings a new assumption that cannot be ignored or rejected statistically.

4. Flow by hour during day depends on weather conditions

Concluding the section related to data exploration it is important to remind the effort done and why it was gathering and proving assumptions based on possible correlations between variables.

Understanding our data, it is a step forward to understanding behaviours. Understanding behaviours, they can be predicted. Taking in account these results, further developments will be done following described assumptions and mechanism will be made to predict them.

3.1.3.2 Data Outliers

At this point, independent variables are already found and proved statistically. It was gathered a set of base assumptions where analytical models can rely and be built on. Before starting to build analytical models, there are however some concerns that are need to be overtaken. It is well known the existence of factors that can hardly deteriorate predictions results. Outliers are one of them, since out of range values can have a big impact on correlation models where analytical models are build one.

It is described as outliers, a value which falls out of the set results that are regularly obtained. Once an outside of range value is obtained, this value can be analysed follow two contradictory assumptions.

1. Value received means that some unusual is opening, and on streaming analyses, this is a value which it is pretended to be captured.
2. Value received means a missed or wrongly captured value and it needs to be discarded.

For analytical building models proposes, the assumption which will be relied on is the last described in order to build the most accurate model discarding any possible out of the box values. The first assumption must not, however, be dismissed. On a real time, processing engine, one of the things demanded to be outstand, are the non-regular values. A streaming based approach based one first statement will be build and explained on *Chapter 4.2.3 - Speed Layer Data flow***.**

Following the assumptions that it is being worked on with an uniformly distributed dataset following the normal distribution, once standard deviation is calculated, out of the range values can be extracted through different outlier detection mechanisms. Based on dataset characteristics, and the continuous properties of data, the chosen method will be relied on interquartile range calculation. The theorem is presented on Equation 1 - Outlier theorem.

$$[Q_1 - K(Q_3 - Q_1), Q_3 + K(Q_3 - Q_1)]$$

Equation 1 - Outlier theorem

Outlier detection theorem was chosen instead of others as it can be calculated on a cluster distributed manner and as it can be easily managed between batch and stream calculations. Subject will be described carefully further on chapters *4.2.2 - Analytical Layer Data flow* and *4.2.3 - Speed Layer Data flow***.**

3.1.3.3 Analytical Models

Stepping through on a volatile and demanding market, a static interface containing past results is not suitable or acceptable anymore. Understanding the past, is no doubt the first step to improve the present, however, more than ever, it is mandatory to be ambitious and have a look on future.

The path taken so far will determine the successful of the next step. Having the data been extracted, transformed and load, the independent fields identified and the outlier values extracted, all conditions are gathered to analytical models can be built and next requirements accomplished. The goal and applications behind the building of one analytical model, can vary by business scenario. For this specific case scenario, the goals will be:

1) Recover lost data during gathering stage
2) Predict future results by a given hour
3) Predict probability of total park places to be complete

Defined the goals, and the means to achieve it, all it is there left is how to put things together. As described on *Chapter 2.4.2 - Data Analytics***,** there exists a significant amount of algorithms with analytical capabilities. They all are very specific and when not correctly chosen, expected results accuracy can be seriously compromised. The search criteria of the algorithms for the described use cases will be based on analytical models properties category. Analytical models categories will rely on input data properties and output expected result.

Regarding expected values to predicted, a model can be built on regression or classification models. Recalling *chapter 2.4.2.2*, regression models are need to be based on when continuous values are the required output. Other way, classification models would best fit on models when a discrete value is need to obtained. On the two firsts case scenarios, continuous values are expected to be obtained and further, and so a regression model is need. Left, input data properties would also need to be identified. Given, the number of independent identified variables a regression problem can be characterized as binary or multiclass. As referred before, it was identified several independent variables having impact on data results. Determined more than one independent variable, a multiclass algorithm would need to be use. Data linear property is the last major characteristic to complete the analytical model group identification. Based on data quadratic properties, a nonlinear algorithm was need. Summing up, based on input data identified characteristic a nonlinear multiclass regression model would be need to be chosen to be the needs of the two first case scenarios.

From the set of available analytical modes compliant with the identified characteristics, two have highlighted itself. Gradient-boosted trees and Random Forest were the algorithms who best results presented. Random Forest related results can be observed on *Table 1 - Gradient Boosted Tree Standardized Mean Square Error*, and Gradient-boosted trees related results on *Table 2 - Random Forest Standardized Mean Square Error*. Results obtained under the random data split training model, splitting seventy per cent of data for testing proposes and thirty for test purposes.

It1 (%)	It2 (%)	It3 (%)	It4 (%)	It5 (%)	It6 (%)	It7 (%)	It8 (%)	It9 (%)	It10 (%)	avg(%)
0,176	0,159	0,128	0,168	0,151	0,175	0,156	0,174	0,166	0,175	0,162

Table 1 - Gradient Boosted Tree Standardized Mean Square Error

It1 (%)	It2 (%)	It3 (%)	It4 (%)	It5 (%)	It6 (%)	It7 (%)	It8 (%)	It9 (%)	It10 (%)	avg(%)
0,199	0,213	0,213	0,201	0,207	0,193	0,195	0,215	0,229	0,192	0,205

Table 2 - Random Forest Standardized Mean Square Error

The presented results are the outcome of the algorithms models evaluation after a training phase. After models built to both algorithms, under the same test dataset the mean square error metric was calculated. For simplicity purposes, results calculated to a normalized dataset.

The mean square error is a common used metric used to evaluate regression models. At its core it measures the distance of the predicted value with the original outcome. The lower the obtained values the better analytical, once it would be translated on far less distance between predicted and original values. The applied formula can be analysed on *Equation 2 - Mean square error*.

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Equation 2 - Mean square error

Analysing the two models both based on the ensemble of decision tree models, was Gradient-boosted which best average presented. On its genesis, the differences between both can be explained given the test dataset properties. Having between both algorithms the only difference the training process, Gradient-boosted tree fit its best when less level of spanned trees are built opposite to Random Forest. Random forests would be the chosen model on cases with a much larger dataset variation having greater variance when less spanned trees with bigger deep would be need[38].

Through Gradient-boosted the two firsts requirements were fulfilled. The first, recalling, aiming to recover lost data during gathering stage and as a backup to further faults, pretends to have as result the mean value of occupation for a missed hour for a missed day. The missed values were possible to be recovered through the trained model composed by the vector encompassing the formal fields park identifier and hour, enhanced by the discovered correlated features on exploration stage, week day, weather condition and holydays factor. A practical result of the Gradient boost tree regression model application regarding first requirement can be seen on Figure 19. On picture, the underlined yellow represents the recovered data while on context of correctly captured data.

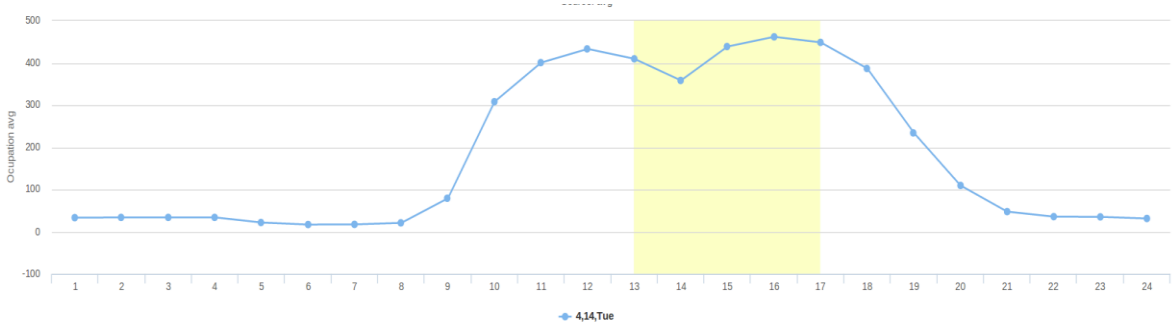

Figure 19 - Lost data recover mechanism application

The second analytical requirement, as already stated, will leverage the same forecasting algorithm. Having the same input parameters, however applied on different contexts, a new predicting model will be created. The input parameters exception will be the feeding hour cease to be the real data hour to be a forwarded hour. Thereby, given the existing values of current hour, a forwarded hour corresponding value will be calculated.

An application of the second requirement can be seen on Figure 20, which from a first given starting hour, the following twenty-three were generated. The red line is the result of applied algorithm where other lines are the illustration of real gathered data for the same provided input characteristics.

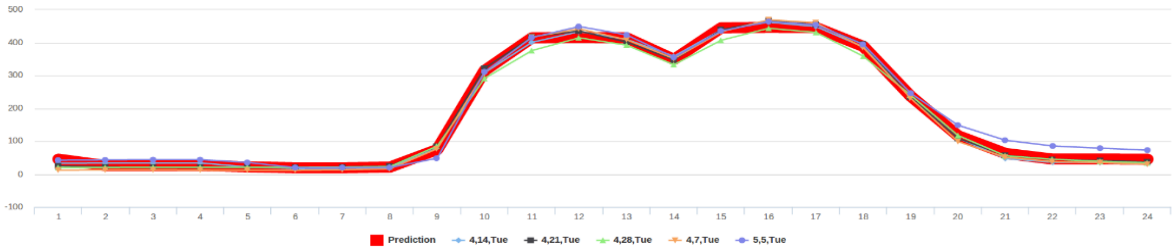

Figure 20 - Predicted data mechanism application

Different from last described requirements, is the last case scenario. While before continuous average values were need to obtained, now a discrete value is expected. Instead of an average value ranged from zero to the max average existing value, now a concrete meaningful value is expected. At a given hour for a given day, based on current results an answer about park full occupation will need to provide. The park will be full or the park will not be full.

Changing the expected values, which is identified as one of the properties to characterized the analytical model category to be chose, also the used model would need to be modified. Given the discrete values, required to be obtained, regression models would not fit as a good approach to be followed anymore. Whereas a label or category is need to be obtained, as this case is, classifications model tends to be used. No other change noticed from the two first described case scenarios, keeping the input data characteristics untouched. Nonlinear multiclass regression category of models will be now shifted to nonlinear multiclass classification models. Selected from the existing libraries repository, based on described properties it stood out the decision tree classifier and Naïve Bayes classifier

Decision tree classifier related results can be observed on *Table 3 - Decision Tree Standardized Mean Accuracy*, and Naïve Bayes related results on *Table 4 - Nayve Bayes Standardized Mean Accuracy*. Results were obtained under the random data split training model, splitting seventy per cent of data for testing proposes and thirty for test purposes.

It1 (%)	It2 (%)	It3 (%)	It4 (%)	It5 (%)	It6 (%)	It7 (%)	It8 (%)	It9 (%)	It10 (%)	avg(%)
0,961	0,958	0,962	0,969	0,960	0,966	0,962	0,967	0,966	0,968	0,964

Table 3 - Decision Tree Standardized Mean Accuracy

It1 (%)	It2 (%)	It3 (%)	It4 (%)	It5 (%)	It6 (%)	It7 (%)	It8 (%)	It9 (%)	It10 (%)	avg(%)
0,879	0,85	0,86	0,859	0,87	0,87	0,85	0,86	0,86	0,87	0,863

Table 4 - Nayve Bayes Standardized Mean Accuracy

The presented results are the outcome of the algorithms models evaluation after a training phase. After models built to both algorithms, under the same test dataset the accuracy was calculated. For simplicity purposes, results were calculated to a normalized dataset.

Different from regression related methods, the metric used to evaluate the cases scenarios were different. Now, not the difference between obtained results and real results will be calculated, but also the correct results. For that propose accuracy, metric was used.

Analysing the two models was the decision tree which stood out. The different relative easy to be explain other than the one existed on regression models is given to the fact of naïve Bayes classifiers strongly depends on features independence, which is not evidenced under the case on analyses.

Worth to mentioning are the relative higher noticed accuracy of the Decision tree classifier. Assertive accuracy led the dataset to be reviewed looking after a higher result explanation. The result, at the end, can be justified by the fact that only a small percentage of parks for the existing sample have its filled occupation equals to park capacity. The result is also enhanced given the fact of when they fill up the previous data behaviour variation is very similar.

Summing up all the road done so far, on Figure 21 presents the overall gone through flow. The data, first gathered from a queue system mechanism, was cross against external enrichment API and stored into an immutable dataset. On a trial-error iterative process, aiming to identify the most influencer independent fields, data was gone through similar processing stages. Pruned out of the range values thought the outlier's extraction with fields of interest identified, analytical models were built. At the end, the obtained results on previous stages would end on much effective data analytical models built with prediction very closer to expecting results.

Analysed and understood the main data existing capabilities, numerous application and flows can derive by. A set of pieces were already built. Based on gathered results and remaining requirements to be fulfilled, next stage will describe the solution architecture.

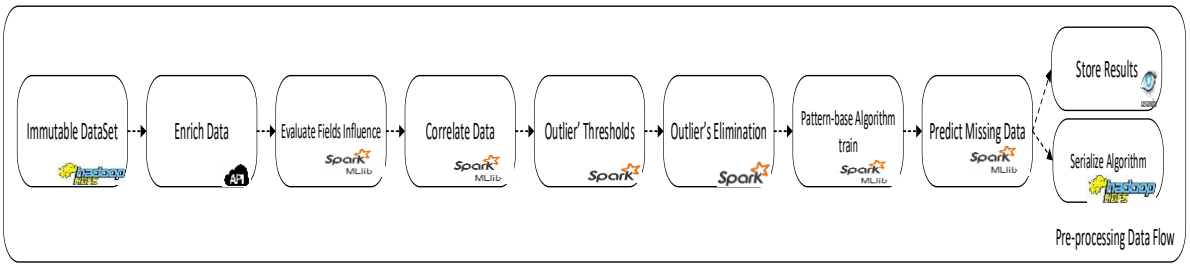

Figure 21 - Data exploration flow

4 ARCHITECTURE

It outlines the approach followed and proposed architecture to address defined needs. All implementation details will be discussed too.

At this stage should be clear the type of data it has been working on and how far it can be explored. Data was processed, information extracted, and insights can already be leveraged. However, all it was built so far was a set of static applications far from the reality that is expected to be reach with this current work.

On this section, new questions will be taken aboard. Taking in account, results previously obtained, the main idea is to carry them and lead it to a final solution capable of fulfil all the described requirements subject of this work. Architectures, technologies and framework available will be carefully analysed and its advantages and disadvantages weighted.

Defined the goals and proved that they can be reached was a first accomplished step, now the best way to put it into a real solution is to put previous results in practice on an end-to-end solution and expand then to all described use cases of this document.

Recalling the described requirements on *Chapter 3.1 - Use case Description*, there are two references using cases that summarize the context of the problems that will be carry around until the end of the work. This requirement, even generalist will emphasise the type of problems will serve as reference to the further developments and solution architecture:

1. The solution, should be capable of provide strong guarantees for historical data and be able to process and query it on an accurate way.
2. The solution, should also be capable of provide real time data insights with low latency.

The first, more than a requirement, a guideline, demands a batch based approach to be use. When present a problem with greats quantities of data which are continually increasing, and forecasts going on the direction to keep the tendency, a distributed and scalable infrastructure is need to be built. Terabytes of data are expected to be received, stored and further processed every day. A straight approach, recalling the product properties discussed on *Chapter 2.4.1 - Hadoop*, would involve a Hadoop or Hadoop like solution. A framework capable of scale horizontally and process on distributed way big amounts of data would perfectly fit for the use case under analyses.

Taking advantage of it fault tolerance distributed file system, the HDFS, data store guarantees would be assured. Additionally, leveraging data locality processing, Map Reduce framework could be used to fulfil the required processing capabilities on a cost-effective way.

The Hadoop, not only as a component also as an ecosystem, could provide all the tools to the described requirements could be successfully achieved. With a single shot solution the hub of technologies could be easily reached as demonstrated on Figure 22.

Figure 22 - Overall bulk processing requirements

Following the second requirement, also a guideline, a completely different approach needs to follow. A real-time based component must to be into question. With the goal to obtain real time insights, wherefore profitable information can be extracted, data must be processed while still on flow. Where Hadoop fits its best for batch processing, Storm or Spark Streaming fits them for real time processing. Instead of bulk processing throughput, on flow processing capabilities are mandatory in order to latency can be diminished and meaningful results obtained.

Opposite to first guideline approach where the data hub could be easily composed by one straight technology hub, streaming processing technologies are only concerned on process the data and not on make them available. The solution will have the necessity to be enhanced with an extra query piece. An ordinary database could fit, depending the data queries needs. At extreme, a distributed database could be easily extended. The Figure 23 demonstrates how the pieces' arrangement could be done.

Figure 23 - Overall stream processing requirements

Analysing the requirements by its own, summing up, a bulk distributed processing approach would need to be leveraged to fulfil the main guideline and its underling use case requirements. A completely different approach is needed to be taken in order to solve the second main guideline. A streaming based approach is mandatory to use.

Combining the two main guidelines together is needs to use two different approaches. Aiming to reduce the number of pieces in use, for solution building simplicity and complexity proposes, it was made an attempt to reuse the same approach under the both guidelines needs. The conclusions are present below.

Following the first bulk processing approach on both guidelines, data accuracy, integrity and availability are preserved but latency is strongly compromised. Once, there is a real need to obtain real time insights this can no longer be an option.

Boarding the second real time approach on both guidelines, real time information could be gathered and first gap approach completed. However, data accuracy, data drill down granularity searches, or even data reprocessing needs due to human error for bug fixing or features enhancement would be compromised. Additionally, analytical needs could be set aside given the no existing analytical stream support.

Back to ground zero, there are two completely different requirements, and no one size fit all solution. Relaxing requirements would solve most of described drawbacks and eventually resume both needs to one only approach. Dropping requirements is not however a solution. The two main guidelines would have to end up accomplished, and a way to cross both real time and bulk solution was need.

With the goal to solve similar problems, Lambda architecture was first built. Designed fit on distributed architectures, it takes in consideration the both bulk and real-time processing requirements. Lambda architecture will further be the referenced architecture to extend where main developments will be relied.

4.1 LAMBDA ARCHITECTURE

The lambda architecture is based on a set of strongly reliable concepts. Designed by Nathan Marz it foresees the most common needs of distributed systems. Resuming the main requirements of current work, Lambda architecture attempts to balance required latency enforced by the need of real time results with the bulk process throughput and fault tolerance. At its source, it foresees the building of three different components: Analytical Layer, Speed Layer and Serving Layer. All elements components together can be visualized on Figure 24.

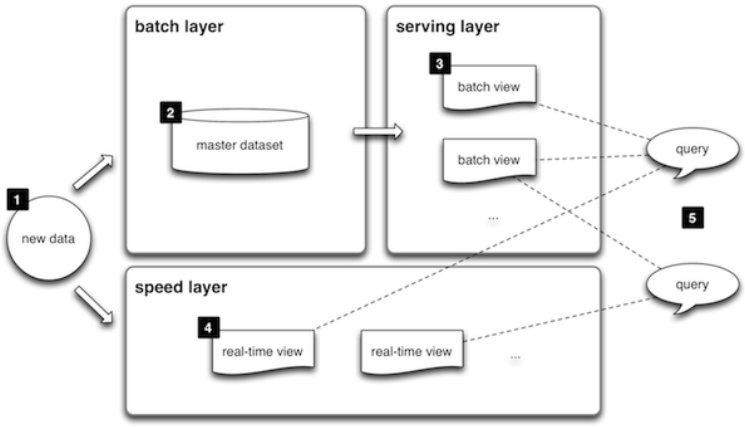

Figure 24 - Generic Lambda Architecture [39]

The Analytical Layer is the reflection of the bulk processing requirements. The build of this component is justified by the need of the existence of a distributed processing mechanism capable of access immutably stored data and process it on a scalable way. Both processing and central storing mechanisms are under the context of the analytical layer.

At this layer, there are enforced two characteristics strongly advised by the author. The first is concerned to the data repository. An immutable and append only data store should be chosen. Also characterized by the literature as Data Lake, its creation envisions the raw data reprocessing needs further when new futures are added or bug fixing is need to be done.

The second concerns to data processing idempotency. Not only the chosen framework but also chosen algorithms should be idempotent. Assuring idempotency, data can be reprocessed several times on different sorting orders or contexts, and, at the end, the obtained results are the same. At is source it is enforced the use of the functional programming paradigm.

The Speed Layer reveals the real time processing requirements. In order, analytical layer latency can be compensated, a streaming processing mechanism capable of process data while in flow is a required component.

A static bolt similar to the previous described Data Lake, which open periodically and trigger the processing mechanism, it is clear that is not an option on a streaming context. A transporting queueing mechanism should be the source of the input data to the streaming processing mechanism.

At last, the Serving layer is the layer that will formalize how the components will be integrate and how should the previous Analytical and Speed layers operate tuned.

It is foreseen by the author that the distribution of source data by the two underlying Analytical and Speed Layer. As consequence, also similar results should be obtained at the end of the processes. Processed data in both layers should not only be combined, but also available to access by external mechanisms.

The idea, behind all these three layers, even that seems contradictory to everything said behind, is to maintain the focus on, and only on the real time requirements. The results prevenient form the streaming layer are the results which will first feed the serving layer.

The results coming from form the analytical layer, can be seen as backup results which periodically will accurately be calculated and replace the previous calculated results by streaming layer. The reason of results overwrite are explained by the outlined bulking processing layer requirements creation at begin of *Chapter 3.1 - Use case Description*.

Summing up, real time results will be computed through the Speed layer and available at serving layer ever since. Analytical layer results will be, only periodically, publish at serving layer. Bulk processing accurate results will replace the batch time calculated results.

Published results on serving layer will be mostly required metrics and refined results only. The results storing mechanism should be compliant with the data access patterns and will only further discussed. Although it was given to understand, Analytical and Speed layer requirement results will not so linearly be the same. Illustrated are the main concepts under the Lambda Architecture that resemble the current work requirements. Underlined concepts will be reutilized and further implemented.

Nonetheless, the highlighted concept will need further to be adapted to current needs. At the end, a new architecture will further need to be build and extend. The *following* Figure 25 presents an overall diagram of the followed architecture to assist the final solution. Each component will be described later on forwarded chapters and technology adoption justified.

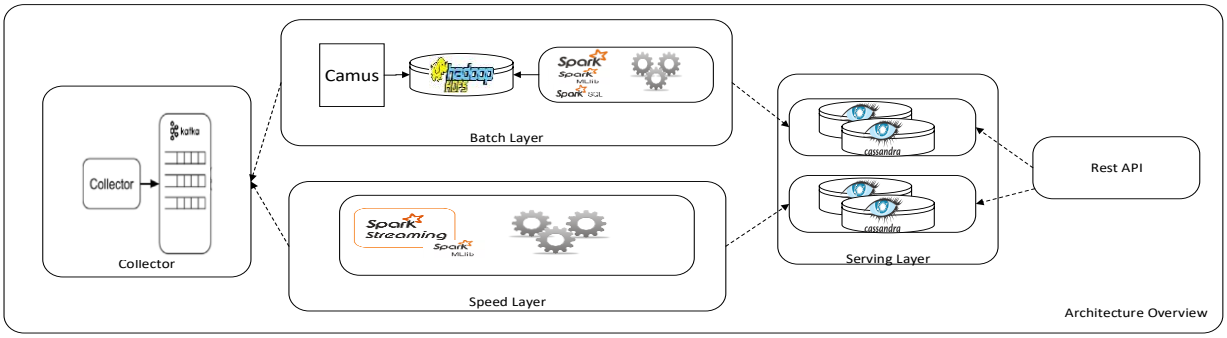

Figure 25 - Extended and customized Lambda Architecture

4.2 Implementation

The following section, will describe on a thoroughly manner, the developments made to accomplished the current work proposed goals. Take into account all the covered landscapes after a long analyses period, the road turned to be clear. All the gathered insight and results will bring to solution design and implementation.

Given the solution amplified magnitude, a complex set of tools were need to be gathered and properly install and configured. For simplicity and management proposes, required software and tools were installed under two similar environments. A first experimental environment, with the development tools, was created for development proposes. For analyses and testing, a second environment would end up to be created.

Sharing the same set of tools, dividing them it is the configurations modes. For simply developments proposes, on the development environment all configurations were done under a standard standalone mode. Ambitioning further goals, the configuration mode of the test environment was done under a pseudo-distributed mode.

The difference between the installation modes is, while on a standalone mode, the tools and software are running on an isolated single unit, on the pseudo-distributed mode,

even under the same processing unit, several different isolated containers are virtualized and a distributed environment is simulated.

The required tools and software listed below resulted on a set of considerations that will further be explain during the following section. All versions were matched together in order to conflicts be avoided. All used tools and software are resumed below:

- Ubuntu server 14.01
- Oracle Java 1.8
- Scala 2.10.4
- IntelIj (only on dev. environment)
- Apache Hadoop 2.4.1
- Apache Spark 1.2.0
- Apache Cassandra 2.1.0
- Apache Kafka 1.2.0
- RabbitMq 3.4.3
- Apache Zookeeper 3.5.0
- LinkedIn Camus
- Glassfish Server

Following the best developments practices and principal, for code maintenance proposes, all code will be stored under a private repository BitBucket[40]. All developments and configurations will be available on demand.

Following the Lambda Architecture requirements, developments were divide thought the described layers. Adding to the existing layers, Analytical, Speed and Serving Layer, will be a Collector component. This latest added component justified by the queuing mechanism need announced by the Speed layer requirements, will be separate to a new logical division since it will be reutilized also by Analytical Layer.

4.2.1 Collector Data flow

A mechanism capable of bring data from network and distribute it to the further pieces of the solution is required. A feeding component needs to be use as the trigger to the following processing units. None less, the component has to be horizontally scalable.

Queuing mechanism built around described proposes are the component that best fits for the current needs. Studied and analysed on *Chapter 2.6 - Data Brokers*, some important queueing characteristics stood out. Highlighted also by the Lambda architecture guidelines follow are resumed the most critical characteristics need to be gathered on a broker mechanism.

Reliability, under the current context it is present as one of the most important requirement. Recalling, all data will need to go through this component before forwarded to following processing layers. Faced with an unreliable queueing mechanism, under a failure scenario, all data will be simply lost.

Fault-tolerance semantic is another of the most highlighted requirements. Under a failure scenario, when the systems suddenly partially go down, a tracking mechanisms following data life cycles is need to be guaranteed assuring data delivery. Even expecting further processing systems to be completely idempotent, there will always be the need of incremental implemented metrics to implement that will turn on miscalculated metrics when data delivery semantics is compromised.

Following the underlying needs, a mechanism already existed with the goal to transport the data over the network built around RabbitMq technology. RabbitMq is a messaging broker that provides an intermediary system for to safe place the messages until they were successfully delivered.

By making use of an asynchronous exchange mechanism, data exchange properties would be guarantee and no message would be lost as it is surrounded by a variety of reliable recover mechanism.

A first application was written leveraging it, and followed approach was tested into the gathering data stage already described on *Chapter* 3, aiming to collect the first sets of data to further be analysed and to support the base of the current work.

Characteristic of the queuing mechanism are the different transporting segmentation mechanisms. The existing system did not only forward the data under the described needs, but also other different types of data. Built around the topic exchange pattern, through a filtering key unnecessary data was pruned.

After a first gathering period, missing series of data were notice. It was realised which RabbitMq reliable control mechanisms were not configured and as the transport system was out of the scope control, being use under other proposes and applications, properties could not be modified. Under the current scenario, missing data is not an affordable possibility. Other solutions were required to replace the existing mechanism.

Once a new solution is required to be chosen, then with the goal to diminish the adaptation plan to further pieces of the solution, the new queuing mechanism should at its source support the different technologies of the delivery processing layers. Being batch layer system built around HDFS, and Speed Layer on Spark Streaming as it will be identified and explained on later sections, a capable mechanism of transport data to both sources with slightest adaptation process was preferable. Between a wide range of offer, Kafka and Flume queuing mechanism were the system that highlighted itself.

The Flume, built around sources and sink pattern mechanisms, provides a channel to a reliably transport data between two systems. Among its advantages, it supports direct interaction with HDFS through a specialized built Sink. For other hand, Flume connector will only be available on Spark Streaming 1.6 and a costume connector would have to be implemented.

Kafka, built around a producer and consumer pattern, guarantee the exchange of messages in categories similar to RabbitMq topics. Unlike Flume, it is supported by Spark Streaming in its genesis. On other hand, data transition had to be made by a third party application. Camus supported by its genesis by LinkedIn created on based proposes, would presents itself as a top candidate.

After studied and vantages disadvantages weighted, Kafka was the option that stood out. More than full support required to move data to HDFS, is the full support for transmit data to the streaming base solution once HDFS will operate under the Analytical layer, an offline layer.

Additionally, Flume was originally build to move big amounts of data choosing throughput over latency where Kafka fits its best propose for the real time Seep Layer. On Figure 26 is provide and overall picture of Kafka that will be used and put in practice on the solution.

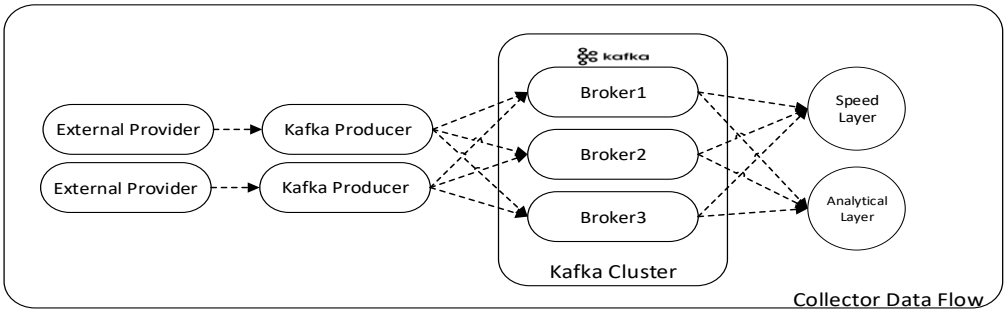

Figure 26 - Collector data flow

4.2.2 Analytical Layer Data flow

The analytical layer is the representation of the bulk processing requirements. Outlined, were already the needs that led this specific layer to create. Nonetheless, before implementation process can be described and technology adoption clarified, this layer creation factors, which will also be the requirements to fulfil, will be recalled and reinforced under the current proposed work.

The real-time requirements imposed by the need of most updated results to be gathered and measured, was what lead the two difference processing units to create. Recalling, there were a set of capabilities, which even could be reach through on flow processing, they would overload the processing unit and strongly compromise the data accuracy. There was other that could not be accomplished.

It will be based on these requirements that the Analytical Layer will be built. More than on the leftover requirements it will be the workhorse of the solution and the reason why the latency will be reached on Speed Layer

Addressing these goals means not only described properties that are assured on solution, also relaxed on Speed layer. Dismissing data semantics, backward data processing, data reprocessing accuracy and analytical model training from Seep Layer, the solution is not get only richer but also more robust.

With the goal to overcome all the described requirements, underlined by the current work case studies, and also enforced by the Lambda architecture principals, the Analytical Layer construction process will be focus under the building of two main components, where it fits the master data repository and the scalable processing engine.

The first, a central repository capable of store all incoming raw events prevenient from the collector stage required. Immutability, is the main principal to be followed when constructing a master data Lake. Received data needs to be stored as it arrives untouched. Guaranteeing, data immutability, under an-append only dataset, further exploration processing needs, new featuring application, or even bug fixing will be possible to be achieved.

Not only, there are other characteristics needed to be provided by the chosen data storing component. Following the transversal solution needs, it needs to be scalable and therefore distributed.

Recalling, all data will be stored under this mechanism, making it a single point of failure. Reliability and availability mechanism should at its source by supported or at least allowed through backup third party mechanisms with no further adaptation plans needed.

At last, accessing patterns should be taken into account. This component will support always under bulk processing, big amounts of data from a source point, and at the end to a delivery point.

All described requirements are need to follow, and not a single one can be dropped behind. When not supported at its genesis, workaround components would be need to be built.

Sharing the highlighted concerns, built around similar proposes it found the Hadoop File System. Described on *Chapter 2.4.1 - Hadoop*, all the properties are thoroughly follow.

To the described requirements, supported by HDFS is the fact that it can be configure over commodity hardware. Being part of a compete Ecosystem, a set of tools can be easily integrated and when leveraged properties such data processing locality and network optimization can be enhanced.

Not only has the HDFS extensibility leaded it to be the fair choice. A determinant factor was the simplicity and clarity that it was build around. Beyond being a filesystem, a high-level interface is offered to manage the most common used features. The tasks are easily achieved based on Linux like commands. As an example, on Figure 27 is the instruction representing the needs of a Hadoop directory creation.

```
9
10  Hadoop fs -mkdir /hadoopdir/hadoop
11
```

Figure 27 - Hadoop directory creation instruction

HDFS was embraced as part of the solution, and immediately was installed and configured on a pseudo-distributed mode. The configurations applied to the installed and configured distributed environment, and the form how it will be leveraged will deeply rely on the adaptation mechanism that will need to be built to move data from the collector stage.

As presented on last section, there is no straight mechanism to deliver the data from Kafka component to the HDFS. A third party data transition mechanism will need to be adapted to the current solution so data can successfully upload from the collector.

A new adaptation component will not however need to be built. Under the same proposes an already existing process will be reused.

Camus[41], originated from LinkedIn is a data ingestion framework built to manage the loading of great amount of data from Kafka to HDFS on an incremental way. Through a set of pre-configured Map Reduce jobs, the Kafka data will be handled and fetched to HDFS. Configurable at it source, custom modifications are not only allowed but also encouraged. Taking advantage of this, a set of features were leveraged in order to fit required goals.

The first feature to be enhanced was the data dispose date. It is configured by default, which from a given Kafka topic, the data written to HDFS by its arriving order. Recalling, the current context is propitious to late data arriving. Based on M2M data, when a network mechanism is overload, the data could not arrive on the proper order.

This fact takes importance once the bulk processing mechanism under analysis will run on offline mode under a range of provided dates. Taking advantage of default features would lead the incremental metrics to be miscalculated.

A custom process following the data topic timestamp field would need to be followed. Required modification were done under the context of the *com.linkdin.camus.coder* class extensibility.

By extending the outlined class, custom Kafka data reading related modification are allow to be applied. Similar, by extending the class

com.linkdin.camus.etl.RecordWriterProvide data to HDFS writing mechanism could be enhanced.

Another enhanced, the last main word to be mention was done at the compressing factor level. Taking into consideration the fact that the data will be read under ordinary circumstances only once, a compressing factor would allow the save of data storage.

Not all, other custom modifications are allowed to be leveraged. Camus on its own configurations it is not activated by itself. In order it can be triggered a background periodically job would need to be built. Even if it could be easily overcome, thought Cron jobs[42], as it would be done later, the problem would be the schedule time period. Whereas only by time period the Camus process can be activated, a considerable amount of small files can be generated.

Recalling HDFS specificities, it fits its best when handling big lots of data, and small files can present a few drawbacks. Camus Sweeper, is a Camus project extension which propose to handle this step back. Camus Sweeper similarly to Camus, is a Map Reduce job which goal is daily collect hourly produced small files into a big final file. Using this option, instead of 24 small files generated per day, a compact file would be generated encompassing remaining files. Given the controlled test environment where it has been work on, the generated files weight is considerable and this option wasn´t implemented. On real case scenario seconds thought would need to be taken.

In order to complete the Camus adaptation process, a real example of how data is pushed from Kafka, and organized under the HDFS, is provided below. After received a considerable quantity of data under Kafka collector mechanism, the Camus Cron job was triggered. For period of time between 20-05-2015 at 8:15 until 20-05-2015 at 8:55 it was created a file containing all the data under the directory described on Figure 28.

```
12
13  hdfs://hostname:hostport/rhadoop/rcoimbra/topics/replicated-topic-data/hourly/2015/05/20/08/ replicated-topic-data.2.0.3.6.1432134000000.deflate
14
```

Figure 28 - Camus directory behaviour

Stepping through the logical process chain, after data loaded to the immutable dataset, it will need further to be processed and explored in order to require metrics and results can be obtained.

Moving further to the processing stage, the second and last component outlined under the Analytical process layer, an extensive distributed framework capable of process big quantities of data is required. Recovering some of the non-taken features by the Speed Layer, there are a set of enhancements that are need to be accommodate by the chosen processing framework.

The data writing semantics, not provided by most of real time processing frameworks is a concern that needs to take. Also the backward data processing, discarded at Speed Layer, are need to be possible to be accomplished. The framework to chose should have the capacity to access vast quantities of data and order it on a cos-effective way. The vast access to all existing data is enforce by the data reprocessing needs foreseen to this processing unit.

At last, data analytical capabilities are need to take into account. Reinforced by the online analytical requirements, offline data training modelling should be allowed so further models can be reused on Speed layer. Mixing the described requirements with the characteristics of Hadoop Map Reduce, a full match is done. Even more, having the Hadoop distributed file system already under use the processing adaptation would easily be overcome.

However, given the experienced difficult caused by the low level offered API and principally the job chaining configuration and implementation verbose need also referred on same chapter, an extension of the Hadoop Map Reduce framework was needed.

The already set of technologies identified were weighted under the faced challenges. Technologies such Crunch, Cascading, or recently Spark and Flink were analysed. In common, they are all build on Map Reduce, and offer a high level API with an enlarged set of third party libraries. Between the highlighted options, it was chose the one that fills better the follow requirements, the Apache Spark.

Spark is a parallel processing engine that uses the same methodologies empowered by Map Reduce. It does not only offer a Map Reduce API, but also improve it internal Hadoop Map Reduce process.

As described on *Chapter 2.4.1.5 - Spark*, Spark is empowered by memory, reducing the I/O needed before, optimizing intermediate between Map and Reduce stage steps in memory. With Apache Spark data locality continuous to be leveraged guaranteeing that data is processed where it is stored avoiding unnecessary exchange of data over the network – one of the most highlighted features of Hadoop. Additionally, Apache Spark provides its own analytics library - MLlib. Stepping through the option Mahout the third party ceased to be need in contrast if Hadoop Map Reduce was chose.

Recalling which two different layers will be built, one based on batch processing, and other on streaming processing, and that processes between both are need to be implemented using different paradigms, a framework which have the capability to be extended to both layers with be a plus.

Between it owns libraries, Spark encompasses also a dedicated Streaming processing library. Through Spark Streaming library, Spark can be extended to the Speed layer, diminishing the developments needs to implement it and number of different frameworks

in use – Spark Streaming developments will be described on *Chapter 2.5 - Real time parallel processing systems*.

Chosen the framework subject of further processing developments, the implementation processes will be finally described below. The implementation processes done by following the already proposed architecture on *Chapter 4.1 - Lambda Architecture* under the Figure 25.

Highlighting the processes under focus on Analytical layer, it also creates the Figure 34 for simplicity proposes. Following stages will by making use of Spark processing framework, explain the transforming, enrichment, outliers, metrics calculation and analytical process.

Recalling *Chapter 3.1.3 - Data exploration*, similar processes have already been explored. Ever possible, same mechanism and methodologies will be reuse.

- Data ETL

Through an input directory path defined by the data periodicity of Camus scheduling processing, all the underlined files, composing the specific batch will be first load.

After data load to the custom RDD, the data Spark processing abstraction, required fields under analyses are possible to extract. From a wide range of identified fields on *3.1.3.1 - Data Correlations*, the field of interest were extracted and transformed into RDD, Spark data abstraction as outlined on *Chapter 2.4.1.5 - Spark* in order they can be manipulate. The manipulation process can be visualized on Figure 29.

```
83
84      rdd.map(line => Record.parse(line))
85
```

Figure 29 - Batch transforming mechanism

- Data enrichment

There are well known the field that have influence on obtained results. Weather condition and non-weekend days, are fields of influence that have impact on results. In order they can be leveraged on analytical models, from outside sources they need to be extracted – external public API. The fields' enhancement process can be visualized on Figure 30.

```
382
383     .map(line => line, weather.forecastCondition, hollyd.getHolidays)
384
```

Figure 30 - Batch feature enhancement

- Outliers extraction

Outlier extraction component was reuse also from exploration results stage. The outlier extraction mechanism is divided into two distinct stages.

On first stage, the upper and lower bounds are defined and extracted thought the outlined mechanism on *Chapter 3.1 - Use case Description*. After achieved the first goal, obtained bounds will further need to be confront with all dataset. Those with values greater than upper bound or lesser than lower bounds will be identified and redirect in order to be analysed further.

Follow described approach, against all dataset the process will need to be done twice. Only after obtained the outliers bounds, the out of range values can be identified. Taking into account described drawback, the process was redesigned in order to the two stages can be operated independently.

Avoid the two stages to be accomplished every time outliers are need to calculated, the results prevenient from first stage will be stored for further uses. A drawback is, however, identified when the processed leveraged on this way. Outlier's bounds will be updated with last noticed results. The issue, however, easily is overpassed by activating the first process only ever the outlier's bounds are needed to be updated.

Further, having the outlier's bounds identified and stored under an interoperable component, this process can be reuse by other, as it will be under the Seep Layer, or third party mechanism. A representative extract of built code identifying the outlier bounds and its pruning process can be visualized on Figure 31.

```
266
267  seqQTree.reduceByKey(qtSemigroup.plus(_,_))
268  data.join(thresholds).filter(outlier)
269
```

Figure 31 - Batch outlier extraction

- Metrics calculation

A set of metrics were need to be calculated and obtain. Same mechanisms were use from exploration results stage and extended to new further metrics exploration.

Between most notable metrics were calculated:

  - Average, standard deviation and absolute values in different time granularities: hour, day, month and year; Top K parks with higher flow over time and day; and Parks with active flow over time identification

In common, all were built around aggregated function supported by Spark. A representative metric calculation, aiding to determine the number of distinct elements on a given dataset can be analysed on Figure 32.

```
286
287 rdd.groupByKey().mapValues(x=>{val zipRdd=rddExHead.zip(rddExTail).filter(rddDiffHT)
288         zipRdd.size compare 0 match {
289         case 1=>countpos.map(rddCountDiff).reduce(_+_)
290 }})
291
```

Figure 32 - Batch generic metric calculation

- Analytical modelling

Analytical models were already discussed on data exploration process previously. The same algorithms and models were use and reutilized aiming to accomplish the different goals. These goals, more important than need to obtain under the Analytical layer, are need to be gathered under the Speed Layer.

Recalling, analytical models are compose by two distinct stages: training stage and test or predict stage. Since it is not reasonable to think that training an analytical model on a streaming framework can be a far option, similar to outlier's approach, the process will split.

The built models will be serialized in order to further can be recovers under Speed Layer. Even it not being a supported feature under the current Spark version, a workaround process was made storing all models parameters separately with the goal of later be reconfigured when deserialize. The training process enhancement can be visualized on Figure 33.

It was not ever mention before or discussed the fact of all used algorithms being base on trees. Noticeable should now be the option taken. No mattering the training time or training approach, a light model was need to obtain so it could be leveraged under the Speed Layer. Other way, any other algorithms would even providing more accuracy results or more efficiently, overload and turn impractical the use of analytical models under any other context than Analytical Layer.

```
303
304 DecisionTree.train(trainingData, Classification, Gini, maxDepth, fieldN)
305
```

Figure 33 - Batch training mechanism

Summarizing all Analytical processes flow is Figure 34. It is provided and overall picture of how every piece fits on final solution under the Analytical Layer context.

Figure 34 - Analytical flow

4.2.3 Speed Layer Data flow

As described on *Chapter 4.2.2 - Analytical Layer Data flow*, most of the main requirements were already address by the Analytical Layer. Data metrics, outliers and analytical results are already possible to leverage. However, given the specificity of the Analytical Layer based on bulk processing technologies, on past data the results will only gathered due to underling technology characteristics.

The subject under analyses on current work, it is not complaint with the gathering of metrics based on late results. The case scenario is constantly changing and further real time results are need to obtained.

Representing the real-time needs, it arises the creation of the Speed Layer. Enforced by business scenario and Lambda architecture principals, compensating the Analytical processing latency, similar results are need to obtain with data while on flow.

Having the same metrics and results calculated under the same way, even as it is enforced by the Lambda architecture is something was previous analysed and rethought. Supported by the already obtained results on Analytical Layer, having in mind the main goal of layer construction some trade-offs will be applied.

Taking advantage of accurate results calculated on Analytical Layer, several requirements would be relaxed and further extended. Having the guarantee that at the end of day all data view results will be replace, for sure this is a risk worth to be taken.

In order to diminish the processing time and as consequence the latency, probabilistic algorithms will leverage when extracting data results. To the heaviest metrics replacement by probabilistic results based on business case data properties other bold strategies will be enhanced.

Most of the times, received data does not present significant status changes. From a set of records, only those with big percentage related to the same event are worth to notice. By relevance meaningless data will be pruned. Stepping thought degrees of freedom, statistically speaking, it is not necessary to know the results of the last hour to determine

the accumulative results of the current year. Taking that in account, only punctual metrics will calculate discarding different granularities types.

A framework capable of support all requirements is need. Between described existing frameworks two considerable options persist where Storm and Spark Streaming best fits. Both similar in its characteristics, no one overlaps the other. Favouring Storms is the fact of it has been the source of the Lambda Architecture design principals. In favour to Spark Streaming, however, is its recent exponential increasing.

Most of the critics given to the lambda architecture which has even led to alternative proposal creations are the difficult of building two different layers with two different technologies such Kappa architecture. However, being the Spark Streaming the choice to take, both layers with almost the same implementation code could be written and implementation process diminished. Recalling, which there are processes that are need to be reuse and extended such analytical modelling the use of Spark would also diminish the adaptation require plan.

Followed describe reasons, Spark Streaming was chosen. Adopted the framework subject of further developments, the implementation processes will finally be described below. The implementation processes were done following the already proposed architecture on *Chapter 4.1 - Lambda Architecture* under the Figure 25 - Extended and customized Lambda Architecture. Highlighting the processes under focus on Analytical layer, it also created the *Figure 40* for simplicity proposes.

- Data ETL

Between Spark and Spark Streaming extension a new data abstraction exists called DStream. DStream are a set of RDD received on same batch interval. Through transforming mechanisms, DStream can however be transform to RDD when required.

Similar process was used to first filter and parse the received data from Collector as the processes implemented on Analytical Layer. However, instead of directly consumed from HDFS, data will be received instantly from the Kafka provider. A pre-processing mechanism need to be implemented based on a Kafka receiver source. The process hereafter, under similar processes would be reach. An example can be visualized on Figure 35.

```
19
20  stream.map(line => Record.parse(line))
21
```

Figure 35 - Stream transforming mechanism

- Data enrichment

As analytical models built based on enrichment fields, the same fields will need to leverage those.

Similar based mechanism like those implemented on Analytical layer, built to accomplish the process. An example can be visualized on Figure 36.

```
28
29  .map(line => line, weather.forecastCondition, hollyd.getHolidays)
30
```

Figure 36 - Stream feature enhancement

- Outliers extraction

Opposite to previous approach, data thresholds will not be calculated but instead reused. Taking the results obtained on Analytical layer, outlier's bounds will be loaded and leveraged on real time.

A database request, or any load mechanism, cannot be done every time a new record arrives to the system. Millions of records are intended to be received, and besides the obtained result will further have to be shipped to each machine where process is occurring on. Instead, all outlier's bounds will be loaded and transmitted to every cluster node at once. Spark Streaming allow the use of these mechanism through broadcasting variables[43].

A read only variable will be load and a copy cached on each machine. Having all outliers bound charged, the same filter mechanism will be done after received data crossed with its bounds as implemented on Analytical Layer explained *acima.* An example can be visualized on Figure 37.

```
38
39  .transform(data =>{data.join(broadcastVar.value).filter(outlier)})
40
```

Figure 37 - Stream outlier extraction

- Metrics calculation

Most notable changes between layers will be notice on metrics calculation. In order to get continuously the last metrics updated, it need to be confronted with last calculated result on previously processed streaming batches through accumulative operations[44].

Spark provides an ideology of last noticed state updating. This ideology provided by *updateStateByKey (function)* method, allow any provided function to be applied to a previously recorded state. An example will be provided further.

Instead of calculate the number of words on entire dataset ever a new word arrives to the system, a previously known state is stored containing the number of existing words and

an incrementing function to last know state is applied. This ideology will be applied on all metrics calculation mechanism.

However, it is not not the only change that will notice. As explained on begin of this section, the heaviest mechanism will be replaced by probabilistic algorithms aiming to reduce the processing time through approximation calculations instead of accurate results.

- Average, standard deviation and absolute values by hour

Taking advantage of the accumulative ideology presented before, an example of metrics summarization can be visualized on Figure 38.

```
49
50  updateRawEvent(value:Seq[Int],state:Option[(Int,Boolean)]):Option[(Int,Boolean)] ={}
51
```

Figure 38 - Stream generic metric calculation

- Top K parks with higher flow over time

Although it apparently seems an easily resolution problem, as explained on *Chapter 2.5.1 - Probabilistic Algorithms*, calculate the top k occurred elements on big amounts of data environment need to be faced as a serious challenge. Considering on the heaviest metrics, its calculation batch approach will be exchanged by a probabilistic based approach.

Count-Min Sketch algorithm is the chosen calculation method to be applied. As explained, through sub-linear hash structures, the events will be map to frequencies. In exchange of the accurate result, when leveraged, an estimate result will be given.

When applied, expected results will determine the most frequent results in the stream. Recalling which an event record raised when a flow movement occurred, when used, the Count-Min Sketch will reply the top k parks where flow changed most.

Additionally, through accumulative calculations offered by Spark Streaming, the results will be merged and preserved through streaming batch during a day.

- Identify parks with active flow over time

It is considered an active park those who from a pre-defined period of time have at least one flow modification. In order to determine the network or the material deterioration, the active parks are needs to identify. In practice what is expected are the distinct parks that over a time did not suffer any flow modification to be identified. Similar to the last describe metric, a heavy calculation is presented.

Apparently easy to achieve through a simple count distinct calculation mechanism, when confronted with big amounts of data, the calculation mechanism need to be rethought since the required amount of data is proportional to the cardinality dataset.

Hyper Log Log probabilistic algorithm was the method chosen to be putted in practice. Based on uniformly distributed of random numbers observation the distinction estimated results are calculated and further parks without flow calculated.

Additionally, in order to fulfil the second requirement, where the results are needs to obtain across a period of pre-defined time, one extra step implementation method is need. When the period of time to be determined is greater than stream batch period of time, further calculation mechanism is need. Periodic reset accumulators would need to be implemented. However, instead of last proposed method, Spark Streaming already offers abstraction way to implement and achieve desired results. Through window mechanisms[45], a set of data can be retained and leveraged. Making use of this mechanism together with Hyper Log Log algorithm, the expected results can already be extracted.

- Analytical modelling

In order to the next hour, results can be predicted and the parks fulfil probability to be calculated, both regression and classification models will need to be used.

Recalling which same models were already explored on Data exploration processes and sequentially implemented on Analytical Layer, in order to meaningful information can be extracted they are need to be implemented on Speed Layer.

Similar to outlier's extraction approach where thresholds were calculated on Analytical Layer and load on Speed Layer on broadcast variables, the same mechanism will be implemented to deal with analytical models. Both algorithms will be load to broadcast variables which when combined with on flow data, most update predictions will be calculated and extracted.

The predicting mechanism can be visualized on Figure 39.

```
145
146   .map(x=>(x,model.predict(x)))
147
```

Figure 39 - Stream forecasting mechanism

Summarizing all Speed processes flow is Figure 40. An overall picture is provided of how every piece will fit on final solution.

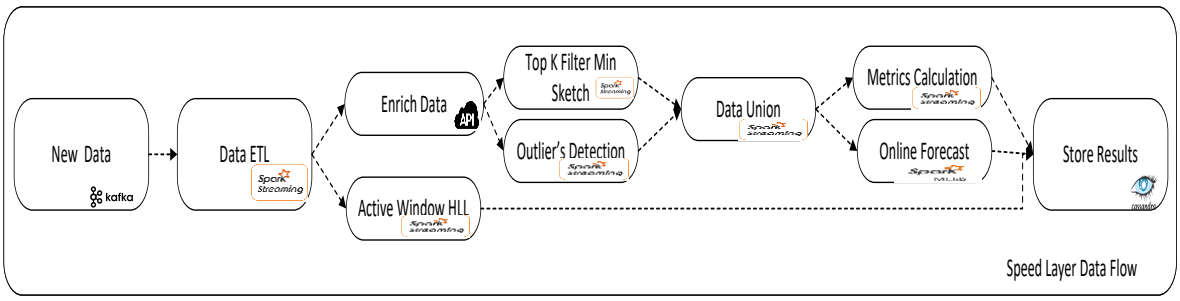

Figure 40 – Real time flow

As presented on last presented figure, most of the existing components were move and reorganized. With the goal to diminish the processing layer latency, Speed Layer experienced some enhancements. Transversal to all components under the current context, from the same source, the data was transformed, and filtered, after gathered. After the first pre-processing stage, however, the data will be redirect into three distinct flows.

A first flow, independent from others, will be in charge of compute the number of parks with active flow. This metric required to be calculate on the use case specification, specifically requires to be evaluated under all received data.

The second flow, the main reason why the fork was created, will through the top K algorithm prune the results by its significance. Under the current context, taken into account, will be the parks that faced greatest occupation variation. Trading of accuracy by latency, the parks with less occupation variation will be discarded from further calculations.

Having in mind all the principles explored so far, when data is discarded even if it presents as meaningless, important insights are always being compromised as explained on Chapter 2.4.2 - Data Analytics. Motivated to hold the most meaningful insights, a third flow was created.

From the initial stream containing all received data, the outliers will be calculated and extracted. Retained outlier's insights will not only be report to the serving layer, but also joint to the results obtained from last flow where after, final metrics and online forecast will be accomplished. The forecasts will, on the same way, retain the information of discarded data rectifying the results by provided insights based on current and past data.

Taking advantaged of created fork, is also Sark internals. Diminishing the overall flow dependencies, the number of independent flows possible to be calculated in parallel increases, increasing also its parallel processing distributed capabilities.

Finalizing the section, important to be highlighted, is that past process compromising data results accuracy can only be implement because at the end of the day, most accurate

results calculated by Analytical layer will be bring to serving layer replacing previously calculated results with less accuracy.

4.2.4 Serving Layer

A set of results were successfully calculated on both Analytical and Speed Layer. At the end through different methods conducted on different flows leveraging different needs, similar results were achieved. On one hand, non-accurate results are produce by Speed Layer, on other same results on accurate way were calculate on Analytical Layer.

Previously highlighted on *Chapter 4.1 - Lambda Architecture*, and enforced as the first goal of Lambda architecture guidelines, the results prevenient from both layer must be brought together and combined.

Non accurate results brought up by Speed Layer need to be present wile accurate results are calculated on Analytical layer. At the end, non-accurate results should be replaced and removed.

With the goal of Serving Layer requirements can be achieve, a repository is needed to store and combine the results prevenient from both sources. Despites all the common goals required on all chosen technology stack, such its capability of handle massive amounts of data on distributed and reliable way, on Serving Layer, a particular characteristic is further required. The data access patter, as already highlighted under the context of the current work, is the finest line between the success and the failure of a chosen data store.

Taking into consideration all operation that would need to be accomplished, from the Analytical Layer, bulk updates are the highlighted access patter to a further data store. From another side, small however on an enormous quantity, inserts are need to be trigger instantaneity.

In common, insert pattern requirements are outlined on both layers. Additionally, given the characteristic of results being time based, an idempotent time series databases with the capacity of making the results promptly available by time would be a plus.

Recalling *Chapter 2.3.2.2 - Wide Column Family Oriented Storage* System*,* Cassandra and HBase, both wide column database are the top candidates and were further analysed. Being both comparable in results matter, Cassandra support and availability enhanced by its best fit of underlined requirements was chosen.

When a NoSQL database is chosen, a trade off properties and features is always made *(Chapter 2.3.2 - Not Only SQL)*. Apache Cassandra is no exception and limitations start right on data modulation.

Opposite to ordinary relation models where at first the model is thought and only after the queries are debated and optimized, with Apache Cassandra on a reverse order the things are done.

Apache Cassandra trades a wide set of non-optimized operation for a small set of offered optimized operations. Offering a short set of operations do not mean that others are left behind but instead, which they need to be considered at modelling stage in order it can be leveraged after on an optimized way.

Based on current work scenario as explained before, queries require to be accomplished are time based and with the goal it can be leveraged on an optimized way, a time series modelling based would be the right model to be followed. Describing the base model that will be extended later for other features is the Figure 41.

Figure 41 - Apache Cassandra Modelling

Base presented model built around a time series model proposes the use of three fields encompassing a row id – which would be the park identifier based on current context, a column value - timestamp when records were generate and the corresponding captured value.

Further fit the model to the business scenario, small modifications will have required to be done. Model will be extended adding new fields to existing column or transforming the value field on a compose data type containing other fields.

Following described approach, enhancing the underlined model, a new row will be created for each different park under analyses and a column with timestamp and value added to the same row ever a new event occurs. All events correspondent to the same park are being concatenate on a row.

Up to two billion columns can be added to a single row[46] that´s, under the current scenario, it can present a drawback that should be considered. With the goal of overpass this constraint, data corresponding to same row id, will further also partitioned by date. Adding a new field to row definition such the time, or month, a new row for each park id would be create by second row definition.

Extending the described model based on the case scenario needs the business data model were built. The most relevant created tables will be described.

With the goal to store the data prevenient from Speed Layer were created 4 tables:

- *m2mDataRaw*:

The table was created aiming to make accessible with one single select statement as presented on Figure 42, all existing results from a given park identifier. It is composed by park identifier, the time when event was generated, the absolute number of occupied places, the hour average of occupied places, the standard deviation, the hourly flow, the number of in cars on park and a flag stating if the record is an outlier or not.

As a first example, the creation syntax following Cassandra CQL[47], can be visualized on Figure 43**.**

```
550
551  CREATE TABLE atnogavit.m2mdataraw (
552  id text,time timestamp, raw double,avg double,std double,count double,flow double,threshold boolean,
553  PRIMARY KEY(id,time)
554  )with CLUSTERING ORDER BY (time ASC) AND default_time_to_live = 1440;
555
```

Figure 42 - Structure creation sintax

```
64  SELECT * FROM atnogavit.m2mdataraw WHERE id ='sMfJISMyfWHgWRIaeCJWtO/8xiwQclsQD5D2yXRmBjJLk/rjVXZESwV0Vw1vkdUz5RBC/xi1Ras0YjmUUmjUOltQdmA/P9' LIMIT 100;
```

Figure 43 - Created structure levarage mechanism

- *m2mDataActive*

The table was created with goal to store the number of active parks at a given park. It is composed by park id, the time when event was generated, the absolute number of active parks.

- m2mDataTopK

Table was created with goal to store the parks with grater affluence during a day. It is composed by park id, the time when event was generated, the absolute number of parks with greater flow during day.

- m2mDataBoD

Given the set of tables already described before, through the creation of materialized views combination, a new set of functionalities could have been make available.

However, materialized views are not available at this current version and a workaround was need to be done just to demonstrate how the obtained results could be extended. On Figure 44, is an example of how it can be done using already available syntax, and how it was done.

```
536
537  CREATE MATERIALIZED VIEW atnogavit.m2mdatatopraw AS (
538  SELECT id,time,raw FROM atnogavit.m2mdataraw
539  PRIMARY KEY (id, time)
540  )with CLUSTERING ORDER BY (raw desc);
541
```

```
CREATE TABLE atnogavit.m2mdatatopraw (
id text,time timestamp, raw double,
PRIMARY KEY(id,raw)
)with CLUSTERING ORDER BY (raw desc);
```

Figure 44 - Materialized views replacement

Example aims to explore the gather of the park with the top occupation average by day which available by the query presented on Figure 45.

```
559
560  SELECT * FROM atnogavit.m2mdatatopraw LIMIT 1;
561
```

Figure 45 - Created structure levarage mechanism 2

With the goal to store the data prevenient from Analytical Layer were created five extra tables:

- *m2mDataRawH, m2mDataRawD, m2mDataRawM* and *m2mDataRawY:*

Based on the same goals that lead to the described table creation on Speed Layer, underlined tables were created on Analytical Layer. The difference between the four presented tables is its time granularity.

As explained at begin of this section the chosen database, does not have the same features available as relation models oriented databases.

Aggregations are one of the unsupported features. As consequence of this, simple queries aiming to determine the overall sum of cars on a specific park by year are not possible to be made. In order to overcome presented requirements, four tables with different granularities were created.

The Table m2mDataRawH was created to store the pre-aggregated results by hour, the m2mDataRawD by day, m2mDataRawM by Month and m2mDataRawD by Year.

On a regular Data warehouse model, the same could have been reach with a single dimension table with a time hierarchy or with technologies such Apache Hive or Impala which both have in common the high aggregation performance.

Support the pre-calculated results ready to be retrieved, further will present significant advantages. Even if it would require pre-data calculation, the retrieve latency would be much lower.

- *m2mDataOutliersBounds*

The last table was created with goal to store the outlier's bounds in order they can be leveraged on an agnostic technology layer. It is composed by park id, the influencer variable week day, hour, weather, holyday, and upper and lower bounds.

Described the idea behind data modelling, it will be presented how it was leveraged. Communication between all described components will be done in two distinct stages. The first with be the data exchange between Analytical and Speed Layer with Apache Cassandra which will be demonstrated above, and the second between Apache Cassandra and Web

Services publishing mechanism which will be described on publishing mechanism topic above.

Facilitating the interaction between described layers and Cassandra, a customer connector already built by Datastax [48] was used. The same interface was use cross both layers. An insert operation is present on Figure 46.

```
311
312         .saveToCassandra("atnogavit","m2mdatabod",SomeColumns("id","time","raw"))
313
```

Figure 46 - Apache Cassandra insert operation through Apache Spark

A read operation can also be visualized on Figure 47.

```
108
109     ssc.cassandraTable("atnogavit", "m2moutliers").select("id","wday","hour","outdown","outup","holl","weather")
110
```

Figure 47 - Apache Cassandra select operation through Apache Spark

Since the data from both Analytical and Speed Layers are gathered and available through a query layer, the results are needed to be combined before presented to final user.

The goal, as described is to combine the most accurate results with the last up to date data. In the end, the combination of results prevenient from both layers will be merged as presented on Figure 48.

Figure 48 - Analytical and Speed Layer Data

As already referred, Apache Cassandra present some query limitation. In order to overcome faced limitations, the data combination between layers will not be done on query layer, but on a higher interface thought the merge between multiple queries.

Before going throw the combining mechanism, first it will be recalled how data is being manage on previous layers.

So far, there exist two sets of tables containing similar results. Having similar data replicated twice on data source, the twice requires resources that would also be required. Deleting unnecessary data is a process that needs to be attending and a purging mechanism needs to be putted into practice.

Recalling which Speed layer is constantly producing insert results, and the Analytical Layer periodically updating it, once a new batch of results is originated from Analytical Layer, those results need to overwrite the ones produced by Speed Layer.

As presented Figure 49, an overwrite mechanism will be done. Given an example of daily update process prevenient from Analytical Layer at 12:00 pm, after process concluded all data processed by Speed Layer last day can already be removed.

Results will not however immediately be removed but after a safety time period set to two times analytical processing need time, ensuring that Analytical Layer processing occurred successfully they will. Under normal circumstances a Cron job would have to be made to constantly delete all date inserted older than two days.

However, Apache Cassandra through a Round Robin like algorithm data expiration period can be defined. Mechanism can be achieved on data model creation trough TTL[49] keyword as described on Figure 42.

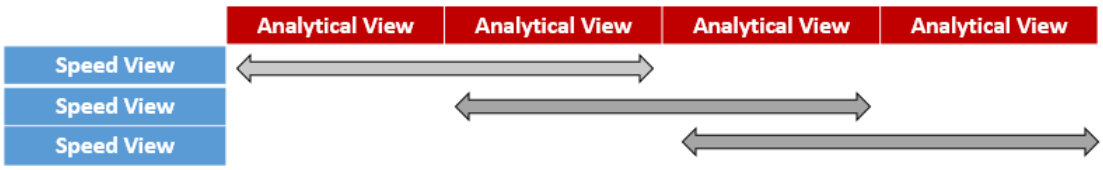

Figure 49 - Speed Layer Views Replacement

Attended the replicated data, all its left is to combine the results. Through crossing query mechanism based on query described on Figure 50, results will be combined.

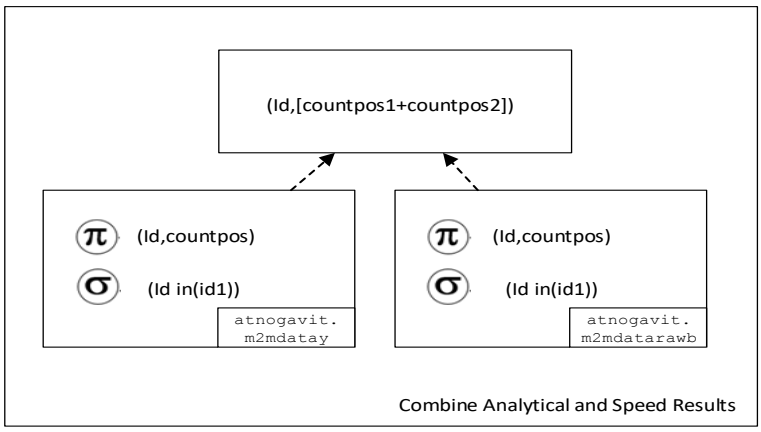

Figure 50 - Analytical and Speed Layer Result combine mechanism

With the goal to make available the results to the end user, that will later be available under a Web Interface, the most accurate data must be combined with the last updated results. The final result will be available under the accomplishment of two distinct queries, one for each Analytical data layer and Speed data layer:

1. Analytical Layer historical data query

Ever result needs information from a period greater than Batch layer processing time, first query will always be made to most accurate data.

From the obtained results, last available date will need to be extracted in order to know ever since the data which is required to be fulfilled with Speed Layer results – Latest date will be provided by last result set record, once data was inserted by its sorted time (Data insert sort mechanism is available through clustering order by syntax as can be observed on Figure 42).

2. Speed Layer Last Update data query

From the obtained results from last query, and taking advantage last know data date, ever since a new query will be made to Speed Layer related tables. After concluded both result sets are need to be join and concatenated forming the last results ready to be presented to final users.

At this stage being results combined, a final stage described on last requirement will be address – Data publishing. With the goal to results can leveraged, they need to be available to be exported to a dynamic built interface. Given the portability offered by it, a Web Interface will be implemented.

Through several patterns, this goal can be achieved. First thoughts pointed that Web Framework should be used to support the developments. However, with eyesight to further results extensions, the developments will be decoupled and made available on first instance through Service Model pattern to be towards available to be consumed by any third party and presented at last.

Combined results will be made available through service endpoints. Web Service were constructed through Java EE support mechanisms. An example of a published endpoint can be analysed on Figure 51.

```
157     @GET
158     @Produces("application/json")
159     @Path("/getM2mDataD")
```

Figure 51 - Java Web Services Publish Endpoint

Later, it is expected which through a simple Web client, exposed endpoints can be explored and data be published dynamically. Web client example made with JavaScript Asynchronous requests is presented on Figure 52**.**

```
127
128     type: "GET",
129     url: "http://database-swat.aws.atnog.av.it.pt:8080/ConnectCassandra/webresources/generic/getM2mDataD",
130     dataType: "json",
131
```

Figure 52 - Java Script Consume API Client

After consumed and transformed, the obtained data is ready to be upload to a final interface. A dashboard will be made with the goal of main results can be presented. From a set of existing libraries, Highcharts was chosen given by its API extensibility and scalability recalling *Chapter 2.7 - Data Visualization*.

Leveraging the pre-built graphics provided by chosen library it was implemented the core functionalities demonstrated on Figure 53 and Figure 54**.**

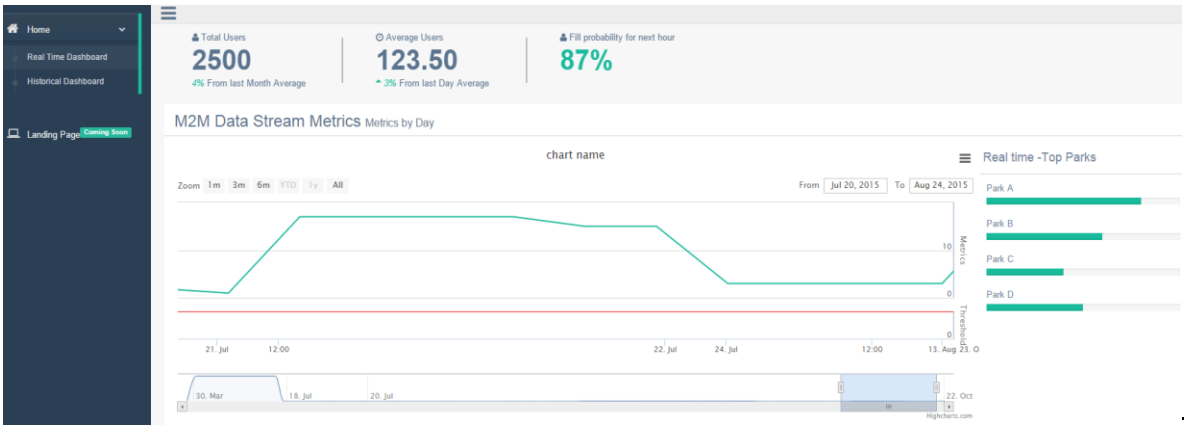

Figure 53 - Real Time Insights Web Interface

On Figure 53, the main page is presented with the most updated metrics. Leveraged only by real time information are displayed and exported a subset of the implemented functionalities:

- Overall real time metrics results

The real time results can be visualized across two different widgets and by the main graphic. The first widget, aims to summarize the accumulator number of places occupied so far under the current moth.

The second widget was made with the goal to summarize the overall average of occupied places by day.

Finally, on main graphic, can be observed when underlined at any point, the absolute and average values regarding to the occupation metrics.

- Top K

The parks with most affluence are also outline and highlighted. On right pane of the screen, a widget prioritizing the most frequency used parks can be visualized.

- Outliers

The outliers are information also possible to be visualized under the main page. Under the main graphic, a red line can be visualized. Ranged between values from zero to one, when faced against with an out of range value, the outlier can be easily identified.

- Predicted filling probability

A widget filled with a probability of the parks location to be all occupied was created. The filling probability will be calculated given the last obtained results against the analytical built models. The probability value is determined by the last obtained accuracy value obtained by last trained dataset.

Not all, similar summing up graphic was built. Based on past data, the graphic presented on Figure 54, aims to present the historical data. The graphic can be accessed thought the left searchable panel.

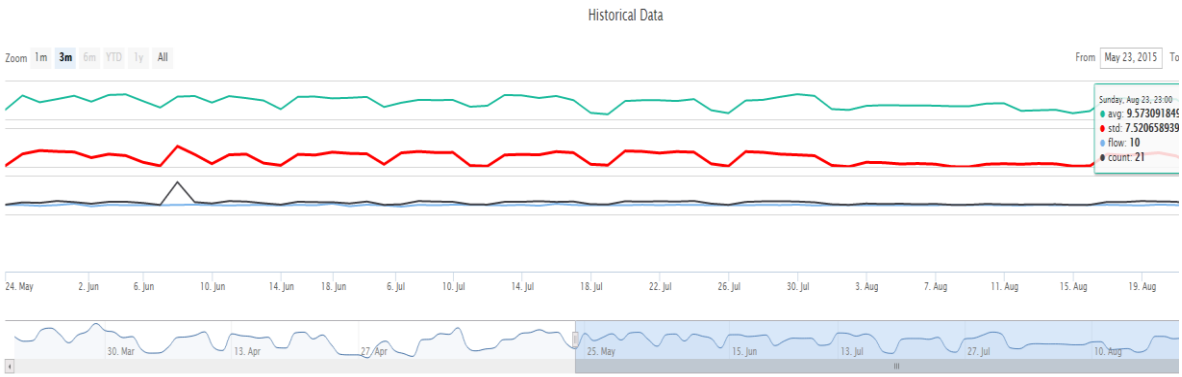

Figure 54 - Batch Time Insights Web Interface

5 EVALUATION

This chapter was created with the goal to evaluate the built solution. Under this section will be putted into test the different parts of the solution and discussed the gathered results

At this point, overall project architecture and its components were already minutely discussed and balanced. Bringing all decisions together, it is reached the time to evaluate the accomplished work. On this *Chapter* different tests will be conducted to the solution on different scenarios. In the end, a clear picture of the major strengths and weaknesses is expected to obtained and discussed.

The selected use cases are based not only on existing real word needs, also mostly on further data environment growing expectations given the big data context of the current work. This way, put into test an extrapolated environment, not only the functional requirements are guarantee but also non-functional requirements - where scalability takes the most important part.

Aiming to produce future data growing needs, existing dataset was incremented with additional generated data. In order to avoid inaccurate test results data generation process was carefully extrapolated strongly based on real data characteristics. Existing dataset was replicated randomly based on real world data properties so that the final result can be reliability used and accepted.

Therefore, to evaluate the different parts of the solution, it was designed three different test case scenarios with equally three different goals.

The first test scenario goal will be to test the solution proof of concept. Based on existing use case scenario needs, described on *Chapter 3.1 - Use case Description*, it is expected which all required metrics could successfully be extracted while the data on the flow and at the end of the day all gathered data could be successfully reprocessed and obtained live non accurate results replaced.

Different from first case scenario where real data were use only, are the following case scenarios two and three where an extended dataset will be use.

The second scenario, rather than the first will be more objective and focussed. Into test, that will be only part of the solution – Analytical layer. The test will consist on the evaluation of the solution behaviour when putted against always increasing amount of the data. Scalability properties analyse will be the main focus of this case scenario.

The third and last test case, similarly to last described scenario will evaluate part of the solution too. Now, it is the Speed Layer that will be put into test. The same, always

increasing data ingestion strategy will be use aiming at the end to measure the stress point of the average amount of data that can be process.

The tests under each of the described test scenario were conducted on a strictly controlled environment. Avoiding volatile memory and cache mechanism to be leveraged, tests were scheduled to be executed during several days at always different days 'time. Supported by Cron tool, all tests were all automatically schedule and conducted.

A job execution plan examples can be observed on Figure 55. The Cron job under analyses was periodically schedule to run every six hours, storing the results to a predefined file. Similar script was written to schedule both Analytical and Speed layer, being the major difference the triggered period and the input path.

```
566
567  #0  */6 * * *   atnogcloud
568  now=$(date); echo " Current time : $now "  > /home/atnogcloud/hadoop/spark-1.2.0/bin/file.txt;
569  cd /home/atnogcloud/hadoop/spark-1.2.0/bin/;
570  ./spark-submit --class AnalyticalLayer.BatchProcess /home/atnogcloud/hadoop/sparkproje/Spark1/target/scala-2.10/Spark1-assembly-1.0.jar
571  hdfs://database-swat-master:9000/user/atnogcloud/m2m/data/parks/* >> file.txt
572
```

Figure 55 - Automated Cron Job

Concluded an overall description of each case scenario and the environment where tests were conducted, bellow it will be detailed each of the described test individually.

5.1 CASE SCENARIO 1

Before any scalability or stress test can be conducted, the system should first demonstrate to be capable of process the real world existing needs where the solution was shaped on. On first test case scenario the system behaviour will be evaluated under a real environment.

Recalling the system architecture and goals, is expected that the solution should be able to ingest and process data while in the flow within the minimum latency possibly causing no congestion. Not only, system should also be able at the end of the day to be capable of reprocess all received data in order to calculate accurate results and replace inaccurate results gathered online.

By processing unit, respecting described requirements and design architecture, case scenario will be sub divided into two sub scenario. First sub scenario will encompass the evaluation of the analytical layer, while the second the speed layer. Both will derive from the collector layer described on *Chapter 4.2.1 - Collector Data flow* . The end test condition is overreached when serving layer is populated.

5.1.1 Speed layer processing scenario

From the ground moment, since the start point of the solution trigger, real and meaningful data will be received. Data while on the flow will be identified through collector layer, which further will be redirected and consumed by speed layer at flow time.

At this point, the only imposed results will be the process of received data while in flow extracting all identified metrics causing no congestion. In order to evaluate test acceptance verdict, it was gathered several results corresponding to processing time taken at randomly chosen times of day. Iteration results were gather measuring the time in nanoseconds. For simply proposes results were round to seconds.

Gathered results can be observed on *Table 5 - Stream real data evaluation results*.

Size (Mb)	It1 (s)	It2 (s)	It3 (s)	It4 (s)	It5 (s)	It6 (s)	It7 (s)	It8 (s)	It9 (s)	It10 (s)	Avg (s)	Avg (Mb/s)
4.05e-4	1,6757	1,6613	1,6354	1,6904	1,6293	1,6268	1,6008	1,6549	1,641	1,6641	1,65	0,00045

Table 5 - Stream real data evaluation results

Under the real case scenario, it was received an average of 0,000405 Mb by second, which were processed in average at 1,65 seconds. Analysing the results, 2 main factors are important to be outlined.

The first, is the expected data to be processed, which it is way under the stress point calculated under the *Case Scenario 3*. The second, it is the enormous average of time to process the given small set of provided data. Even if the second point seems to contradict the first, the difference is explained by the start-up time taken by Spark on every batch. Simply increasing the batch time from 1 second to 2 seconds, even duplicating the received data congestion would be avoided. Minutely gathered results pointing in this direction can be analysed further on *Case Scenario 3.*

5.1.2 Analytical layer processing scenario

At the end of the day, after a complete cycle of data successfully overreached, through the analytical layer, the data will be reprocessed and gathered metrics through speed layer during day replaced.

Following solution architecture, similar to speed layer processing scenario, the data entry point will be the collector layer through the Kafka queue mechanism. However, instead of directly feed an online receiver mechanism, the data will be delivered to Camus.

It is from Camus hourly HDFS generated files that all data will be reprocess. In order to evaluate the viability of the solution, the gathered data were processed ten times at ten randomly chosen period during ten days. Iteration results were gather measuring the time in nanoseconds. For simply proposes results were round to seconds.

Collected results can be observed on *Table 6 - Batch real data evaluation results*.

Size (Mb)	It1 (s)	It2 (s)	It3 (s)	It4 (s)	It5 (s)	It6 (s)	It7 (s)	It8 (s)	It9 (s)	It10 (s)	Avg (s)	Avg (Mb/s)
35	84	85	85	85	85	88	86	84	86	84	85	0,41

Table 6 - Batch real data evaluation results

Analysing the results is possible to conclude which daily gathered data were processed under an average of one and half minutes. Having no comparative results, no conclusive judgements can be obtained, however given the case scenario under analyse, obtained results are easily acceptable. With the goal to more conclusive and objective results to be obtained, given the context of the solution, scalability test would need to be conducted. A deep scalability analyse will be conducted on case scenario 2.

5.2 Case Scenario 2

Built solution has proven thought first test case scenario to be capable of process existing data for the case scenario it was shifted for. However, presented solution since born stage was thought to be scalable and operate on a data extensive environment. Motivated by distributed and parallel processing technologies, it is expected to have the capacity not only of process the provided amount of data but much more extensive amounts of data.

Different from previous test case scenario, in order to allow conclusive results to be contained, a stricter test scenario will be build. The test scenario under analyses was written as an extension of previous test scenario described. Instead of focus all solution as one, leveraging extrapolated generated data out of a real environment the test will be focus only to a specific part of the solution - Analytical Layer.

Under six different dataset composed by different amounts of data bulk processing scalability will be tested. At the end, the throughput of the different cases scenarios will be measure and compared.

Each dataset was processed over a total of 10 iterations. Following the schedule rules described before, the same mechanism was applied to initiate the different tests. At the end the results were gathered and those can be analysed on *Table 7 - Analytical Layer Scalability benchmark*.

Iteration results were gather measuring the time in nanoseconds. For simply proposes results were round to seconds.

Size (Mb)	It1 (s)	It2 (s)	It3 (s)	It4 (s)	It5 (s)	It6 (s)	It7 (s)	It8 (s)	It9 (s)	It10 (s)	Avg (s)	Avg (Mb/s)
313	832	827	822	853	819	818	830	821	851	827	830	0,38
547	1375	1326	1326	1289	1269	1307	1333	1285	1287	1299	1302	0,42
809	2015	1964	2012	2034	1985	2070	1976	1992	2015	2026	2008	0,40
1618	3873	4142	3872	3879	4070	4121	3835	3919	3901	3883	3958	0,41
4043	9817	9825	9831	9799	9889	9913	9773	9871	9816	9830	9838	0,41
8086	19928	19837	19407	19901	19648	19647	19813	19572	19739	19733	19700	0,41

Table 7 - Analytical Layer Scalability benchmark

Gathered the results for each dataset individually, in order to measure solution scalability all results will be shift to a linear graphic in order to a clear analysis can be obtained.

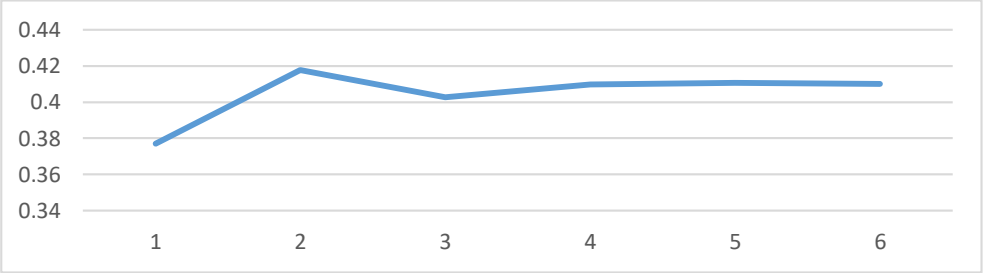

Figure 56 - Analytical Layer Scalability Analyses

On Figure 56 are present the results of the different gathered analyses. Each dataset is represented individually by the horizontal axis values ranged from 1 to 6. Each value is present sequentially given the rows table, where value 1 represents the first row 6 the last. Vertical axis values, represents the range of throughput for each analysed horizontal axis case.

As can be observed, from the analysed sets of data being the first composed by 313 and last 8086 Mb, the overall throughput of solution does not experience major changes. Slightly variations on deeper analyses are however observe between the first and the second and between second and third cases.

First variation can be explained by Map Reduce characteristics. As explained on *Chapter 2.4.1.3 - Hadoop Map Reduce* once exposed to small datasets performance drawbacks can be verified since Map Reduce best fits environments with terabytes of data to be processed.

The second variation between 2 and 3 data sets cannot so easily be explained once it was not expected. Conjectures however can be raised trying to explain this variation. Raised by HDFS number of blocks stored, the number of completely filled bocks can explain the variation. Also since Spark strongly leverages the main system memory, when observing that case 2 process 547 and 3, 809 Mb it can be raised the option between two referred cases, data to be processed left to fit on memory, and even the processing optimized point can be somewhere between case 2 and 3 when the dataset size is balanced with main memory in use.

After case 3, data processing throughput is very similar compared with remains cases, from where can be raised and stated the affirmation which presented solution is perfectly scalable and given results, the processing time can be extrapolated using the noticed slope of approximately 41 Mb per second.

5.3 Case Scenario 3

Similar to the tests executed under case scenario 2, one specific part of the solution will be extensively tested and evaluated. As a last case scenario, is proposed the evaluation of the Speed layer. This layer has already proven itself as capable at first case scenario being capable of process available data on existing constraints. However, in order to exploit it and to be obtained more results it will be evaluated under a different data environment context.

Likewise, to use case scenario 2, the real time layer, will be proven against 6 different sets of ever increasing data. As result of test, it is expected to be taken important insights not only regarding layer scalability but also the layers' stress point.

Tests accuracy has been as described before a subject that has been take seriously. Not different from previous scenarios, not only the same mechanisms will be followed but also extended. In order to best isolate the batch processing time of the stream layer, the risk of a set of data to be split by different batch streams could not been taken.

On Streaming Layer, data is constantly being pushed and several process through continuous windows are being processed. So accurate results can be accomplished, 2 extra mechanisms were implemented.

- Data entry point isolation

So quantity of data to be pushed can be controlled in every stream window, data will be read and consumed instead of from implemented broker mechanism, using file control mechanism thought file consumption. Supported by Spark library is a feature that allows directory monitoring assisting file consumption once it released – *textFileStream* (Directory)

as presented on Figure 57. Leveraging this functionality, through *Cron* jobs, periodically files will be move to a pre-defined directory so system behaviour can be analysed later.

```
88
89    ssc.textFileStream("hdfs://database-swat-master:9000/user/atnogcloud/m2m/data/parkssamplep9/")
90
```

Figure 57 - Data isolation mechanism

- Monitoring approach isolation

Spark Streaming already support in its genesis the monitoring gather statistics. By extending ***StreamingListener*** Class, a set of metrics to be collected ca ne gathered. After defined, through mechanism presented on Figure 58, the defined metrics can be analysed. For test propose was gathered the total processing time plus processing process delay.

```
19    ssc.addStreamingListener(benchmarkStream)
20    ssc.start()
```

Figure 58 - Data monitoring mechanism

Defined data set was processed over a total of 10 iterations. In the end, the average results calculated are present on *Table 8 - Stream Layer Scalability benchmark*. Iteration results were gather measuring the time in nanoseconds. For simply proposes results were round to seconds.

Size (Mb)	It1 (s)	It2 (s)	It3 (s)	It4 (s)	It5 (s)	It6 (s)	It7 (s)	It8 (s)	It9 (s)	It10 (s)	Avg (s)	Avg (Mb/s)
0,117	1,9435	1,9932	1,8175	1,8593	1,8648	1,9686	1,803	1,8845	1,9948	1,9114	1,91	0,06136
0,234	2,0321	1,9936	1,9453	2,0903	2,0222	1,9807	2,0671	1,9178	1,9343	2,0947	2,01	0,11661
0,468	2,2673	2,2026	2,3348	2,231	2,2541	2,3823	2,2713	2,3285	2,2388	2,3715	2,29	0,2047
0,937	4,1535	4,3257	4,2628	4,2996	4,2357	4,1341	4,1048	4,3668	4,2305	4,1016	4,23	0,22164
1,875	8,1545	8,2785	8,6331	7,7291	8,6163	8,054	8,2612	8,8692	8,5754	7,5414	8,28	0,22645
3,75	18,1462	17,5387	18,6512	18,038	18,0672	18,575	18,6287	17,8242	17,9775	17,9817	1,91	0,20662

Table 8 - Stream Layer Scalability benchmark

Summed up the results on *Table 8 - Stream Layer Scalability benchmark* will be for the simplicity of analyses proposes shifted to linear graphic. Below, processing layer measures can be deeply exploited.

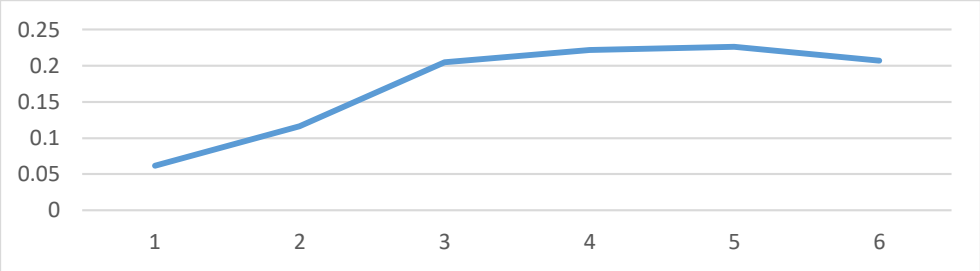

Figure 59 - Stream Layer Scalability Analyses

On Figure 59 are present the different gathered results. Each dataset will be presented individually on horizontal axis with values ranged from 1 to 6. Each number will correspond further to the row number of its representing table. Vertical axis values, represents the range of throughput for each corresponding dataset.

As can be observed, from the analysed datasets being the first composed by 0.117 Mb, and last 3.75Mb, major deviations can be noticed which further will be explained.

Until the processing of the third data unit, the average throughput is increasing almost exponentially. Analysed throughput variation can however be easily explained by the start-up latency which joint with the underlying operations performed by map reduce characteristics when exposed to small amount of data result, causes the reduction of throughput. Recalling, same variation has been observed on test case scenario 2 being however noticed on a smaller scale.

Overcome the smaller datasets, after the third test, data processing unit, the average throughput seems to stabilize. At this point, processing capabilities starts to overlap the noticed slow star-up.

Obtained results, even obtained on a reduced scale data environment, provides good expectations of the adaptability of the analysed layer to a major data amounts environment needs.

6 CONCLUSION

This chapter was created with the goal to highlight the main conclusions gathered during the built of the current solution. It will also outline the future direction to be followed to allow the continuation of the work

6.1 WORK OVERVIEW

On last decades, not only the amount of produced data has changed, but also the way companies look at it. What once considered as meaningless bits, today it has been proved to be the source of insightful information which when leveraged on right context lead to the settlement of the business success rate.

The new bold challenges brought by late changes, overstep all existing technologies and in order to face them, news means are need. Once again, as history has already proven before, the boundaries are overreach when new needs arise. Driven by an ambitious demand, a new stack of product was build.

At a given point, new products started to emerge from all sources. They did not appear however to fit all problems but one very specific problem where it was toward built on. Enforcing the last statement is the notorious example of the existing databases paradigms where properties are given away in order others can be exceeded.

Similar to databases paradigms specificity are the bulk and real time processing technologies which built around very specific trade-offs, when needed to be conducted together, a real set of careful draw offs requirements is need to be carefully balance and measured.

Foreseen under the described characteristics it fits the problem that were face as scope of the current work. Not only the storing, bulk processing or real time processing mechanisms were need to be balanced but also the requirements specificity grounded around data analytics properties which would further needed to be explored.

After a strong research plane overpass, more than on technologies or frameworks, it was realized which through only a customized architecture the proposed requirements could be overreached while the technologies strong and weakness counterweight. Lambda architecture, was the chosen one. Based on underlying principal, both functional and non-functional requirements were address and extended.

Supported by bulk processing mechanisms, analytical layer was build. Through batch processing paradigm, not only the defined metrics could accurately be calculated but also analytical models trained and built.

Justified by the analytical layer latency, with the goal to real time results be obtained, another processing unit were created – speed layer. Upheld by analytical layer, metrics accuracy could be relaxed through statistical algorithms in favour for latency.

Following described requirements, real-time insights were also expect to be obtained based on last gathered real time metrics. Taking advantage of the interoperability empowered by the use of the same source platform under the two different processing units, it allows the reused of the trained models under analytical layer on speed layer.

Taken approach not only show to fit on real environment test case scenario, but also on an extended environment. The success of conducted tests under strict evaluation metrics, can with no dough impose the origin of a robust proposal solution with could be easily adaptable to different use case scenarios where similar requirements are need.

6.2 Future Work

The great thrust given by be bold evolution of existing processing methods and algorithms will lead to more and more data to be process on even more diminished processing times. Driven by the unpredictable outcomes which data processing can lead, an expectable market will always be behind the developments ever pushing for new applications.

Next natural step to take would be the deployment of the solution to a distributed environment. Evaluated under bigger amounts of data, the confirmation of the obtained results would have led, not only to the solution affirmation, as also optimization once mains drawbacks identified.

Given the provided dataset containing real data, other bullet point to be taken further would be the continually data exploration. Even successfully applied on current work, there will be always more external data to be cross against existing dataset.

Recalling one of the most explored ideas of the described work, what at first look demonstrates to be meaningless information, when looked further at the right way, worthless insights can be obtain and random data can gain new mean.

At last, in order to reinforce the current work values, as a future work, a similar solution should be built around different approaches and paradigms. The emerge of a similar developed solution would not lead only to compare the obtained results but also when analysing it cross the taken paths with obtained strength and weakens which in turn would led to the continually increasing capabilities of the presented solution.

REFERENCES

[1] "Internet of Things: online guide to the Internet of Things." [Online]. Available: http://www.i-scoop.eu/internet-of-things/.

[2] "Machina Research's predictions for IoT in 2016." [Online]. Available: https://machinaresearch.com/news/machina-researchs-predictions-for-iot-in-2016/.

[3] "The Internet Of Things Landscape." [Online]. Available: https://techcrunch.com/2013/05/25/making-sense-of-the-internet-of-things/.

[4] "Iot Processing hub." [Online]. Available: http://blog.cobia.net/cobiacomm/wp-content/uploads/2014/05/iot-refarch.png.

[5] "Car Parking Solutions | Smart Parking." [Online]. Available: http://www.smartparking.com/.

[6] "Bosh - Active Parking Management." [Online]. Available: http://www.bosch-mobility-solutions.com/en/connected-mobility/active-parking-Iot-management/.

[7] "Xively by LogMeIn: IoT Platform for Connected Devices." [Online]. Available: https://www.xively.com/.

[8] "Internet Of Things - ThingSpeak." [Online]. Available: https://thingspeak.com/.

[9] "Carriots - Internet of Things Platform." [Online]. Available: https://www.carriots.com/.

[10] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 26, no. 6, pp. 64–69, 1983.

[11] P. J. Sadalage, M. Fowler, and U. S. River, *NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence*.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo : Amazon ' s Highly Available Key-value Store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, 2007.

[13] "Apache Cassandra Write mecahism." [Online]. Available: https://academy.datastax.com/resources/brief-introduction-apache-cassandra.

[14] "MongoDB Blog." [Online]. Available: http://blog.mongodb.org/post/29127828146/introducing-mongo-connector.

[15] "Druid Base Architecture." [Online]. Available: http://druid.io/docs/latest/design/design.html.

[16] "Hadoop Ecosystem." [Online]. Available: http://keywordsuggest.org/375290-hadoop-ecosystem.html.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, p. 29, 2003.

[18] T. White, *Hadoop: The definitive guide*, vol. 54. 2012.

[19] "Apache Hadoop 2.7.0 – HDFS Architecture." [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-

hdfs/HdfsDesign.html. [Accessed: 28-Jun-2015].

[20] "An Introduction to HDFS Federation - Hortonworks." [Online]. Available: http://hortonworks.com/blog/an-introduction-to-hdfs-federation/. [Accessed: 28-Jun-2015].

[21] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.

[22] D. Miner, *MapReduce Design Patterns*. 2012.

[23] C. M. Lewandowski, *Programming Hive*, vol. 1. 2015.

[24] V. Kumar Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O 'malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," *SOCC '13 Proc. 4th Annu. Symp. Cloud Comput.* , vol. 13, pp. 1–3, 2013.

[25] "LA Apache Spark Users Group" [Online]. Available: http://pt.slideshare.net/cfregly/spark-after-dark-la-apache-spark-users-group-feb-2015. [Accessed: 29-Jun-2015].

[26] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Oper*, pp. 59–72, 2007.

[27] "Spark Benchmark." [Online]. Available: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/spark.html. [Accessed: 30-Jun-2015].

[28] "Spark Programming Guide - Spark 1.4.0 Documentation." [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html. [Accessed: 01-Jul-2015].

[29] "Spark_Ecosystem.jpg." [Online]. Available: https://databricks.com/wp-content/uploads/2014/11/Spark_Ecosystem_Chart11.jpg. [Accessed: 29-Jun-2015].

[30] M. Bramer, *Principles of Data Mining*. 2007.

[31] "Probabilistic Data Structures for Web Analytics and Data Mining." [Online]. Available: https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/. [Accessed: 01-Jul-2015].

[32] N. Alon, Y. Matias, and M. Szegedy, "The Space Complexity of Approximating the Frequency Moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

[33] T. S. Jayram, S. Kale, and E. Vee, "Efficient Aggregation Algorithms for Probabilistic Data," *Proc. eighteenth Annu. ACM-SIAM Symp. Discret. algorithms*, pp. 346–355, 2007.

[34] G. Cormode and S. Muthukrishnan, "Improved Data Stream Summaries: The Count-Min Sketch and its Applications," *Fsttcs*, 2004.

[35] P. Flajolet and É. Fusy, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," *2007 Conf. Anal. Algorithms*, pp. 127–146, 2007.

[36] "D3 - Data-Driven Documents." [Online]. Available: https://d3js.org/.

[37] "Highcharts - Interactive JavaScript Charts." [Online]. Available: http://www.highcharts.com/.

[38] “Ensembles - RDD-based API - Spark 2.0.1 Documentation.” [Online]. Available: https://spark.apache.org/docs/latest/mllib-ensembles.html.

[39] “Lambda Architecture » λ lambda-architecture.net,” 2014. [Online]. Available: http://lambda-architecture.net/. [Accessed: 03-Aug-2015].

[40] “Bitbucket.” [Online]. Available: https://bitbucket.org/account/signin/?next=/dashboard/overview.

[41] “Camus-Data Ingestion.” [Online]. Available: http://gobblin.readthedocs.io/en/latest/case-studies/Kafka-HDFS-Ingestion/.

[42] “Cron Jobs.” [Online]. Available: https://help.ubuntu.com/community/CronHowto.

[43] “Broadcast Mechanism.” [Online]. Available: https://spark.apache.org/docs/1.6.1/api/scala/#org.apache.spark.broadcast.Broadcast.

[44] “Accumulators mechanism.” [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html#accumulators.

[45] “Window Operations.” [Online]. Available: http://spark.apache.org/docs/latest/streaming-programming-guide.html.

[46] “Time Series Data Modeling.” [Online]. Available: https://academy.datastax.com/resources/getting-started-time-series-data-modeling.

[47] “Cassandra Query Language.” [Online]. Available: https://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html.

[48] “DataStax CQL Connector.” [Online]. Available: https://spark-packages.org/package/datastax/spark-cassandra-connector.

[49] “Cassandra TTL Mechanism.” [Online]. Available: https://docs.datastax.com/en/cql/3.1/cql/cql_using/use_expire_c.html.