Wind turbines Big Data Proof of Concept

Pau Alarcón Cerdan

Abstract– The amount of data generated in the world is increasing in size day after day, also generating new problems in storing, processing or querying. The world of industry and the energy sector in particular, is being totally involved in this problem, forcing companies to find new alternatives to the traditional ones. This document describes the completion of a Big Data Proof of Concept, using the data produced by the wind turbines of a company in the energy sector, which became interested in initiating a digital transformation, in order to replace its current data management system for a Big Data platform built on a distributed system based on Hadoop. The system was designed following the different stages of the data chain, composed of the Ingest, Infrastructure and Exploitation. Finally, the results were successfully delivered to the company, showing its capabilities such as potential, scalability, latency, storage capacity or heterogeneity.

Abstracte– Les dades generades a través de la tecnología está augmentant en tamany día rere día, generant també nous problemes a l'hora de emmagatzemar-les, processar-les o consultar-les. El món de la indústria i concret el sector energètic, s'està trobant totalment involucrat en aquest problema, obligant a les companyies a buscar noves alternatives de les tradicionals. En aquest document es descriu la realització d'una Prova de Concepte Big Data, utilitzant les dades produïdes pels aerogeneradors d' una companyia del sector energètic, que esdevingué interessada en iniciar una tranformació digital, per tal de substituir el seu sistema actual de gestió de les dades per a una plataforma Big Data construïda sobre un sistema distribuït basat en Hadoop. El projecte ha estat disenyat seguint les diferents etapes de la cadena de les dades, composta per la Ingesta, Infraestructura i Explotació. Finalment, es van entregar de manera satisfactòria a la companyia els resultats obtinguts, demostrant les seves capacitats tals com el potencial, escalabilitat, latencia, capacitat d'enmagatzematge o heterogeneitat.

Keywords– Big Data, chain of data, Hadoop environment, wind turbines, anomaly detection.

1 Introduction

Big Data is a relativity new term, used for collections of large data sets that include structured, semi-structured and unstructured data; from different sources as sensors, devices, video/audio, networks, log files, applications, webs, social media, etc; and in different sizes from terabytes to zettabyte, which normally it is defined using the four V's – high Volume, Velocity, Veracity and Variety. [1]

Using traditional data management systems is difficult to process Big Data, appearing problems more frequently now than ever before, since the data stored over the years are producing delays and limitations on processing, storage, and access to this data.

In order to overcome that limits a Big Data platform can be created, by using a collection of computers connected over a network and managed under software which allows to organize, process, and query all the data stored over a distributed file system. Although this system could be much faster than traditional ones, it is also very expensive and difficult to manage, and more problems that makes it not always a good option to for companies to invest in.

Therefore, companies which have sufficient information to manage have to make a choice , whether keep using their traditional systems and machines, which brings many limitations, or in contrast, invest in Big Data & Analytics with the objective of optimize the time tasks to achieve and business advantage [2].

1.1 Motivation

The aim of this paper is to describe the implementation of a Big Data solution, assigned to Oesia Networks from a company with a strict privacy policy situated in the energy sector that became interested in investing in a new way to treat their data.

Currently, the client company has several wind tur-

• E-mail: pau.alarcon.b@gmail.com
• Specialization: Information Technologies
• Tutor: Ramon Martí Escale
• Supervisor: Roser Avilés García
• Course 2017/18

bines responsible of producing renewable energy, in different wind turbine parks situated all over Spain. Each turbine has a sensor installed into their system, which captures all the possible information generated. Afterwards, this information is saved in hard disks, that contain a huge amount data continually increasing. The company uses a spreadsheet as data analytics engine to make statistics and visualize the results, but as the size of the data grows, the latency of getting results also increases.

Therefore, a Proof of Concept (PoC) was realized to demonstrate to the company that using a simple Big Data solution all the information generated and stored for these sensors could be used better and faster.

2 OBJECTIVES

The objective of this PoC was to show the potential and advantages that could bring a Big Data platform. Having in mind that a PoC is a small or an incomplete project, and because of the strict data privacy policy of the client company, they only provided us with one year data of a single wind turbine park.

Then, the project was planned in terms of the principal objectives required to achieve the final client delivery, following a methodology that is based on the data chain, representing how the information travels through the system, from its generation, until the representation. [3] It is usually defined by several different parts, but in this case it was generalized in three modules, which also represents the three main objectives purposed:

1. **Ingest**
   The first objective of the data chain is to retrieve the structured data out of data sources and storage it in HDFS (Hadoop Distributed File System).

2. **Infrastructure**
   The second objective is to create the application's core and accomplish the ETL process through completing its sub-objectives (Extract, Transform and Load).
   Extract– The objective of the extract is to retrieve the data from HDFS to the tool that will be used to perform the processing.
   Transform– The objective of the transform is to convert the extracted data from its initial structure to the structure that it needs to be loaded into the database.
   Load– The Load objective is to store the data from the tool used for the transformation, to its corresponding database

3. **Exploitation**
   The third objective corresponds to the last module of the data chain, which consists in use the data to find a new value for the company. It includes two sub-objectives, the analytics and visualization.
   Analytics– On the one hand, the objective of the Analytics is to implement an anomaly detection technique based in quantile calculus.
   Visualization– On the other hand, the objective of the Visualization is to create different representations grouped in dashboards or views, using the data stored in the database.

4. **Test**
   Last but not the least, different unit tests related with (scalability, performance, heterogeneity , security, latency time...) has to be done with the objective of achieve the acceptation and satisfaction of the client company.

3 STATE OF THE ART

Each Big Data case, has to be studied and designed as individual one, since there is a range of different frameworks to use, which normally there is one that fits better than the others. Since the proof of concept was focused on the batch execution of a structured dataset, the tools that fit better and were available in the Cloudera 5.12 distribution installed in the Oesia cluster were chosen. The selected tools are listed in order of use throughout the project.

-Cloudera: Cloudera (CDH) is an open source platform distribution based on Apache Hadoop, it also contains a proprietary Cloudera Management Suite which provides management services such as automate the installation process, displaying in real time the nodes and services state, etc.
-Hadoop: Hadoop is framework based in MapReduce programming model, that supports the processing and storage of extremely large data sets, based in a distributed computing environment.
-HDFS: Hadoop Distributed File System or HDFS is one of the Hadoop's core components that stores data on commodity hardware, providing high-performance access to data across highly scalable and fault-tolerance Hadoop cluster.
-YARN: Yet Another Resource Negotiator or YARN is another one of the Apache Hadoop's core components responsible of allocating system resources among various applications and scheduling tasks to be executed over the Hadoop cluster nodes.
-Spark: Apache Spark is a cluster computing technology, designed for fast large scale data processing. Taking advantage of its distributed in-memory storage for high performance processing across a variety of use cases, including batch processing, real-time streaming, and advanced modeling and analytics. Which it has support for running on Hadoop Yarn or on Local.
-Hue: Hue is a browser that allows users to easily search, glance and perform actions on data or jobs in Hadoop clusters.
-Hive: A scalable, fast and extensive Data Warehouse framework build on top of Hadoop, that processes structured data by translating its HiveQL (SQL) queries in MapReduce jobs.
-Impala: Another framework for Data Warehousing which uses MPP (Massive Parallel Processing) SQL query engine for processing large structured datasets for faster access to the data in HDFS compared to other SQL engines such as Hive, Pig or Mahout.
-Spotfire: A Business Intelligence tool for visually analyzing the data that enables to create interactive dashboards, which depict the trends, variations and desity of the data in form of graphs and charts.
-Kerberos: A network authentication protocol designed to

provide strong authentication for client/server applications by using secret-key cryptography.

4 Cluster Specifications

This section shows and explains the infrastructure that was used to carry out the project. The Oesia owns an on premise cluster composed of 6 nodes that uses the Cloudera 5.12 Distribution based on Hadoop (CDH).

On the one hand, the Cloudera packaging and tarball information [4] includes most of the technologies listed in section 3 (Cloudera, Hadoop, HDFS, Spark, Hive and Impala). But the last two of the list (Spotfire and Kerberos) were not included in Cloudera distribution, reason why they had to be installed and configured manually in local.

On the other hand, in Table 1 there is the most important hardware specifications of each node (Cores, Memory and Disk).

Table 1: Hardware specifications

Name	CPU	Memory	Disk
hvi1x0194	12x2.20MHz	96GB	3.5TB
hvi1x0220	12x2.20MHz	96GB	3.5TB
hvi1x0256	10x2.20MHz	108.2GB	6TB
hvi1x0257	10x2.20MHz	108.2GB	5.9TB
hvi1x0258	10x2.20MHz	108.2GB	5.9TB
hvi1x0259	10x2.20MHz	108.2GB	6TB

5 Methodology

In the next sections of this paper are explained the different modules of the data chain (Ingest, Infrastructure and Exploitation), all of them following the same logical structure (Analysis, Design, Implementation and Test).

The Ingest module is the smallest one, and in it the data was distributed along the cluster's nodes. Then, the Infrastructure module, the core, which is structured as an ETL process. Finally, the Exploitation module, used to find new value and insights for the client through Analytics and Visualization processes.

The software development methodology chosen was the incremental model [5], which combines the elements of the waterfall model, with the iterative philosophy of prototyping. So the project was done in a sequential order, because neither the infrastructure could not be deployed if the data had not been extracted before, nor could the visualization be displayed until the data had been processed and loaded. In conclusion, each part required the previous ones to be developed and the project was not finished until it satisfied all of the objectives.

6 Ingest

The objective of the Ingest module is to migrate the company's traditional system to a distributed one. Its Analysis, Design, Implementation and Test are described below in the following subclauses.

6.1 Analysis

The company which this PoC is based on owns 9 wind parks in Spain, with an average of 10 wind turbines each one. The input data come from two different sensors installed in each wind turbine, one that generate alarms and other signals. All of these data are currently stored in a RDBMS, but the company did not give any details of its structure, and only provided us one year of a single park data into different CSV files.

These files were stored in the HDFS, using a Data Lake pattern. The data lake is known as a landing area, a location where the data are directly stored from the data sources to the HDFS as raw data (non-defined structure). It also could be used as a history, contributing a safety factor to the system.[6]

6.2 Design

The Ingest process is represented in Figure 1. At the left side, there is the traditional system, where the data is managed by an RDBMS and displayed through Excel visualization tools. The arrows represents the data flow direction, that goes from the left to the right part, representing the HDFS distribution along the 6th node cluster.

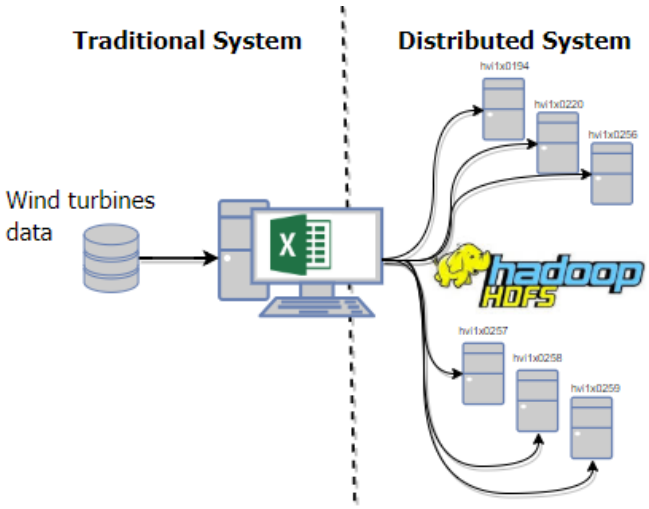

Figure 1: Extraction Design

6.3 Implementation

First of all, the two CSV files with the wind turbines information received from the company were stored in the local file system of the "hvi1x0194" node. Then, the Hadoop framework was used to distribute all the information along the Hadoop datanodes. This process was done using the following command executed in the hadoop shell that copies a file from the local file system, to the dataLake path in HDFS: *"hadoop dfs -copyFromLocal"* with two arguments, the local source file path *"file://home/oesia/pau/data"* and the destination hdfs directory path *"hdfs://hvi1x0194:8020/user/oesia/pau/wtpoc/landing"*.

When the file is distributed along the different hadoop nodes, it is also partitioned in blocks of 64MB, each one twice replicated to increase the fault tolerance, using the default Hadoop configuration from *"/etc/hadoop/conf/hdfs-site.xml"*.

6.4 Test

Theoretically, if the process would had failed, the error would have shown in the Hadoop shell, anyway, it was checked in order to assure that the file was updated correctly:

- Checking the Hadoop logfiles at *"/etc/hadoop/logs"*
- Checking whether the file is stored in HDFS, via HUE (A framework used to data management and visualization), or using the hadoop shell.

7 Infrastructure

The infrastructure is the the core module of the PoC. It is structured as an ETL process (Extract, Transform, Load), and its Analysis, Design, Implementation and Test are explained in the next subsections.

7.1 Analysis

The ETL process is mainly performed using the Apache Spark framework [7].The Spark execution engine allows the creation of a high level application that consists of a driver program that runs sequentially the Extract, Transform and Load modules, executed through in-memory parallel operations on the cluster.

Scala was the programing language chosen, since Spark is built on it and this combination offers the best performance, allowing to access the lasts greatest features that are first available on Scala and then taken to other languages such as Java, Python or R.

7.1.1 Extract

At this point, the raw sensor's data are already stored in the data lake on HDFS. Therefore, these would be extracted again, but in this case it is taken from HDFS and imported to Spark as a Resilient Distributed Dataset (RDD). The RDD is the primary abstraction and the core data structure of Apache Spark. It is an immutable distributed collection of objects, divided into logical partitions which may be computed on different nodes of the cluster. Once the data have already been partitioned as RDD format, the Transform process could start.

7.1.2 Transform

In the Transform process, the data is processed, modeled, cleaned and enriched for a future Load to the Data Warehouse. The module is divided in three different objectives:

Normalize
To start with the Transformation, the Normalize process assures that all the values are in the correct form.

Formalize
The Formalization is the process in which the data structure is modified in order to have the same format as the database tables, where they will be finally stored. Initially, the input file had the data of different wind turbines in the same row, which it had to be transformed using RDD operations, to have a single turbine data for each row.

Enrichment
In the Enrichment process the dataset is enriched by adding new fields for each input event. Two different enrichments were done, the first one calculates the interval of time between two consecutive events, and the second one adds a location field to each event.

7.1.3 Load

This is the last module of the ETL process which starts once the data was normalized, formalized and enriched. Hive, the software used to perform this storage was chosen between other data warehouses, because it is a framework well integrated to work with Spark. The data were stored as a Parquet file, because it was the only format which allowed Impala to access these data, using the Hive metastore.

7.2 Design

The ETL model is a pattern used in most of the Business Intelligence architectures, but each Big Data problem requires a different approach. Figure 2 shows this schema which the data flow is represented from the data lake Extraction to the Load to the data warehouse.

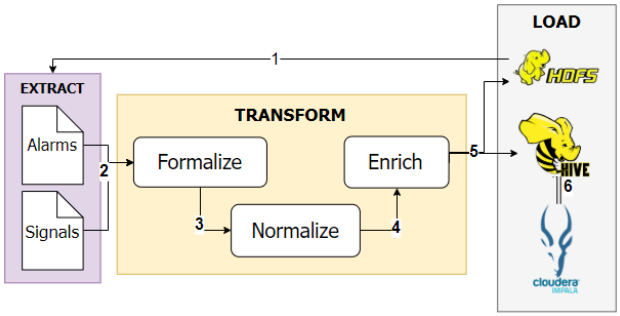

Figure 2: Extraction Design

Each arrow represents a different data operation which drives the information to the next process, with a number assigned to all of them that is explained below to further understanding of the whole Infrastructure process.

1. The data extraction, from the data lake to a Spark RDD.
2. Normalization, the first transformation process where the data is cleaned.
3. The Formalization process that changes the structure of the initial dataset through RDD operations.
4. The Enrichment, the last transformation which adds new value to the data.
5. The Load is executed from Spark, which connects to Hive to store the data already processed.
6. Finally, is represented the relationship between the two data warehouse frameworks used, an automatic process which allows Impala to access to the Hive metastore.

7.3 Implementation

The implementation section deeply explained technical details about each ETL module.

7.3.1 Extract

This process extracts the information from the the data lake path in HDFS, and parse it as Spark RDD. To achieve this, it is necessary to set up the SparkContext [8], with its configuration used to select the cluster resources that the Spark job will use. Then, once created the SparkContext their inner functions can already be used, in this case was used *textFile* as *SparkContext.textfile("hdfs://path/to/datalake")*, which automatically creates the RDD from the sensor's data located the data lake path, this is executed two times, one for alarms and the other for signals.

7.3.2 Transform

In the transformation the data is modeled through the Normalize, Formalize and Enrich processes.

Normalize

The objective of the Normalize process is to clean and normalize the initial dataset by checking the whole fields. Therefore, these are three examples of mistakes that were found and corrected:

1. The first error found was the temperature signal, which sometimes came in bad format.

$$\boxed{^{o} \rightarrow {}^{o}C}$$

2. The second error found occurs when the alarm sensor has communication errors. Theoretically, the alarms indicator would have to come as IntegerType in range of (0 to 3), on the contrary, it came as StringType. Therefore, in order to solve this, a table with a different unique Integer number for each different string received was created. The conversion can be seen in Table 2.

Table 2: NORMALIZED PARAMETERS

String	Integer
Comm Fail	-1
Bad	-2
Failed	-3
Other failure	-4

3. Last but not the least, each signal value was cleaned, since the separation between the real number and decimals was made with a coma ",", but to use it as DoubleType, it had to be replaced by a dot ".".

$$\boxed{11{,}234 \rightarrow 11.234}$$

Formalize

The aim of formalize process is to change the initial data structure and transforms it to have the same structure defined in database tables. Subsequently, in order to further understand this process, following there is a comparison between the initial and final structure.

The initial structure is shown in Table 3, where each row is identified by unique timestamp with an interval of 10 minutes. This initial schema allows us to easily see the information structured and identified by the timestamp, nevertheless, this structure does not work because each park could have a different number of turbines, and each turbine a different identifier.

Table 3: INITIAL STRUCTURE

Eolic Park				
ID	**Turbine1**		**TurbineN**	
Type	**S1**	**S2**	**S1**	**S2**
30-03-2017 02:00:00	12.34	1.45	-21.55	35.11
30-03-2017 02:10:00	46.33	153.34	21.65	-13.67

Therefore, it results as a non-scalable structure which in order to solve this problem, the data were transformed as shown Table 4.

Table 4: FINAL STRUCTURE

Eolic Park			
Time	**ID**	**Type**	**Value**
30-03-2017 02:00:00	Turbine1	S1	12.34
30-03-2017 02:00:00	Turbine1	S2	1.45
30-03-2017 02:00:00	Turbine2	S1	-21.55
30-03-2017 02:00:00	Turbine2	S2	35.11
30-03-2017 02:10:00	Turbine1	S1	46.33
30-03-2017 02:10:00	Turbine1	S2	153.34
30-03-2017 02:10:00	Turbine2	S1	21.65
30-03-2017 02:10:00	Turbine2	S2	-13.67

To undertake this transformation, it is used a map function that iterates over each row, and then for each Turbine of the initial structure. Then, each new row has only one signal value by adding a new field that indicates the signal type. However, it involves the replication of its corresponding timestamp, aero and park, which means that if the initial table had X number of rows identified by the timestamp, now the final number of rows are the different number timestamps multiplied by the different number of signal type fields (S1,S2,..,SM) for all the Turbines (Turbine1,Turbine2,..,TurbineN) in the park. This new structure greatly increases the table size, by replicating information, although it was necessary to store the dataset in the structured table.

Enrich

Once the data was formalized, each turbine is enriched with a new feature that will also be used in the analytics module, adding new value to the data.

1. *Interval of time*
   The interval of time is the time passed between two consecutive events generated for each wind turbine, and it is calculated for both alarms and signals. This enrichment is very useful since (as explained in the second example of section 7.3.2 - Normalize) there could be transmission errors because of the communication or the turbines state, which means that the events generated do not always have the

same interval of time. Table 5 shows how this interval is calculated in an example using two turbines.

Table 5: INTERVAL ENRICHMENT

Turbine	Timestamp	Interval
Turbine1	30-03-2017 02:00:00	0
Turbine2	30-03-2017 02:00:00	0
Turbine1	30-03-2017 02:10:00	10
Turbine2	30-03-2017 02:30:00	30

2. *Location*
   Each event generated was identified by its turbine and park id. Then, using the two fields a multiple column join between the (alarms, signals) tables with another table with the (turbine, park) location coordinates (latitude, longitude) was done.

7.3.3 Load

The Load is the last process of the ETL model, which at this point the data are already in the correct format (Normalized and Formalized), and it is time to load the data into the Hive tables, done through the following steps:

Creation of Hive tables
The first step was to create the a Hive table for each alarms and signals data, using the same schema as the data already formalized in the Transform process. These tables are easy to create with the Hive shell service, in this case was accessed by the HUE UI. Figures 3 and 4 show the structure of the alarms and signals tables respectively. As it can be seen, these tables have a similar structure, sharing many fields as park, turbine, interval, timestamp and day. The different fields are the indicators from the alarms table (yawState, activeAlarmNo, operationState) and the (signal, value, unit) from the signals table. In order to optimize the queries, the two tables were physically partitioned:

Partitioning:
Both tables are partitioned by the *day* field, this will determine how the table rows will be stored in the HDFS, in this case all the rows having the same date will be stored together, in the same directory, which it optimizes the queries by the day field, only taking the respective portion of the data.[9]

```
CREATE  TABLE wtpoc.alarms
  (park String,
  aero String,
  yawState DOUBLE,
  activeAlarmNo DOUBLE,
  operationState DOUBLE,
  stopped BOOLEAN,
  interval DOUBLE,
  timestamp TIMESTAMP)
PARTITIONED BY (day STRING)
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
```

Figure 3: Alarms table definition

Finally, the properties inserted at the last line is used to SET the table as Parquet format. These must be used in order to well share these tables with Impala.

```
CREATE TABLE wtpoc.signals
 (park String,
  aero STRING,
  signal STRING,
  value DOUBLE,
  unit STRING,
  interval DOUBLE,
  timestamp TIMESTAMP)
PARTITIONED BY (day STRING)
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
```

Figure 4: Signals table definition

Parse RDD to DataFrame
Secondly, the RDD which has the information already normalized, formalized and enriched, has to be parsed as Dataframe. A Dataframe is a Dataset organized into named columns conceptually equivalent to a table in a relational database. To create this Dataframe is necessary the Spark SQL package. [10]

Load
Last but not the least, once the table was created and the RDD was converted into a Dataframe, it is time to load the DataFrame to the Hive table in Parquet format. In order to do so, a connection from Spark to the Hive Service is created using a HiveContext object, which receives as parameter the SparkContext created before.[10]
dataFrame.write.mode(SaveMode.Append).format("parquet")
.partitionBy("day").saveAsTable("wtpoc.signals")

7.4 Test

The Infrastructure module was tested for its performance and scalability, creating a benchmark of the whole executions done using different input sizes and changing the Spark configurations.

Scalability & performance benchmark
A benchmark table was made in order to compare the results obtained in terms of scalability and performance by different executions. Table 6 shows for each row, the job id "ID", the input data size "S", the Spark configuration parameters selected (Executors "E", Virtual Cores "C", RAM), and the total elapsed time "T" that took for each one.

Until this point all the development was done using an initial dataset of 73.8MB, but to test the scalability, the dataset size was progressively increased by replicating the same file until having 14.2GB as input.

On the one hand, the scalability test refers to the first 4 rows of Table 6. On the other hand, the performance test refers to the last 4 rows, in which the cluster resources used for the Spark jobs were modified from its configuration. In addition, all of them were launched by setting the master as "yarn-client", which allowed to use Yarn for managing the cluster resources.

Next subsections show the images captured from the Yarn UI of results obtained in jobs 4 and 7, the two jobs of the benchmanrk table that have the most significant information. For both, it is shown its Tasks table (the different task done time used to finish each task) and Executor's table (technical information such as the number

Table 6: EXECUTION'S BENCHMARK

ID	Conf	S	E	C	RAM	T
1	default	73.8MB	1	2	1GB	2min
2	default	882.3MB	1	2	1GB	3min
3	default	2.3GB	1	2	1GB	21min
4	default	14,3GB	1	2	1GB	-
5	custom	14,3GB	3	3	3GB	67min
6	custom	14,3GB	4	5	3GB	62min
7	custom	14,3GB	3	5	15GB	28min
8	custom	14,3GB	4	5	17GB	27min

cores, the IP executors address, filed or assigned, and the amount of memory Ram used for each executor). Captured from the Yarn UI.

Execution 4

As it can be seen in Figure 5, the job failed at the saveAsTable task (Hive load function), due to an out of memory error. In Figure 8, it can be seen that the job reserved 1GB of RAM and one VCore for its single executor (hvi1x0256), which it failed during its execution. Therefore, the job was launched once agai, this time assigned to the (hvix257), which also failed.

Completed Jobs (9)

Job Id	Description	Duration	Succeeded/Total
8	take at SignalTransformation.scala:32	43 ms	1/1
7	take at SignalTransformation.scala:31	47 ms	1/1
6	take at SignalTransformation.scala:30	46 ms	1/1
5	take at SignalTransformation.scala:20	39 ms	1/1
4	take at SignalTransformation.scala:19	50 ms	1/1
3	take at SignalTransformation.scala:17	37 ms	1/1
2	take at SignalTransformation.scala:16	41 ms	1/1
1	first at App.scala:49	49 ms	1/1
0	saveAsTextFile at App.scala:46	2.2 min	114/114

Failed Jobs (1)

Job Id	Description	Duration	Succeeded/Total
9	saveAsTable at SignalHiveStorage.scala:48	18 min	0/114 (9 failed)

Figure 5: Default configuration tasks details

Executor ID	Address	Status	Storage Memory	Cores	Active Tasks	Failed Tasks	Total Task	Task Time (GC Time)	Input
1	hvi1x0257:34594	Dead	0.0 B / 530.0 MB	1	0	2	66	14.0 m (2.4 m)	7.4 GB
1	hvi1x0256:43750	Dead	0.0 B / 530.0 MB	1	0	2	66	14.0 m (2.4 m)	7.4 GB
2	hvi1x0257:37659	Active	0.0 B / 530.0 MB	1	0	3	64	17.7 m (9.9 m)	7.3 GB
2	hvi1x0256:45950	Dead	0.0 B / 530.0 MB	1	0	3	64	17.7 m (9.9 m)	7.3 GB
3	hvi1x0256:35631	Active	0.0 B / 530.0 MB	1	1	0	2	2.7 m (0 ms)	0.0 B
3	hvi1x0256:35605	Dead	0.0 B / 530.0 MB	1	1	0	2	2.7 m (0 ms)	0.0 B
4	hvi1x0257:32769	Dead	0.0 B / 530.0 MB	1	0	0	1	2.7 m (0 ms)	0.0 B
driver	172.21.40.75:45305	Active	0.0 B / 530.0 MB	0	0	0	0	0 ms (0 ms)	0.0 B

Figure 6: Default configuration executors details

Execution 7

In the seventh execution, the resources used were greatly increased, thus reducing the total elapsed time in 41 minutes (see Figure 7). As it can be seen in Figure 8, with this new configuration, the job parallelized the tasks between 3 different executors (hvi1x0256, hvix0256, hvix0258), reserving 15GB of RAM and 5 VCores for each one. In conclusion, modifying the yarn resources used, the saveAsTable finished successfully, but it keeps being the bottle neck process, taking 26 minutes to load the data from the Spark DataFrame to the HiveTable.

Completed Jobs (10)

Job Id	Description	Duration	Succeeded/Total
9	saveAsTable at SignalHiveStorage.scala:48	26 min	114/114
8	take at SignalTransformation.scala:32	41 ms	1/1
7	take at SignalTransformation.scala:31	45 ms	1/1
6	take at SignalTransformation.scala:30	43 ms	1/1
5	take at SignalTransformation.scala:20	40 ms	1/1
4	take at SignalTransformation.scala:19	40 ms	1/1
3	take at SignalTransformation.scala:17	0.3 s	1/1
2	take at SignalTransformation.scala:16	0.4 s	1/1
1	first at App.scala:49	47 ms	1/1
0	saveAsTextFile at App.scala:46	1.7 min	114/114

Figure 7: Custom configuration tasks details

Executor ID	Address	Status	Storage Memory	Cores	Active Tasks	Failed Tasks	Total Tasks	Task Time (GC Time)	Input
1	hvi1x0257:41775	Active	0.0 B / 7.8 GB	5	0	0	80	2.21 h (3.8 m)	9.6 GB
2	hvi1x0256:42462	Active	0.0 B / 7.8 GB	5	0	0	77	2.29 h (3.7 m)	9.2 GB
3	hvi1x0258:45321	Active	0.0 B / 7.8 GB	5	0	0	79	2.05 h (3.3 m)	9.6 GB
driver	172.21.40.75:41363	Active	0.0 B / 1589.8 MB	0	0	0	0	0 ms (0 ms)	0.0 B

Figure 8: Custom configuration executors details

8 EXPLOITATION

This is the last module of the data chain, whose main objective is to apply analytics using the data already processed and stored in the previous modules, and representing it into the screen. This section is structured as the previous sections (Analysis, Design, Implementation and Test), and also subdivided into Analytics and Visualization.

8.1 Analysis

The core of the exploitation module was done using TIBCO Spotfire tool, that was installed in the client side and implemented the main functionality of the Visualization module.

8.1.1 Analytics

The analytics is the process of examining large and varied data sets to uncover hidden patterns, unknown correlations or other useful information that can help organizations make more informed. In this case, the client company required an anomalous behavior detection over the data captured by the wind turbines signal's sensors. The anomaly detection technique was based on calculating simple statistical methods [11] using Hive functions. Let's say the definition of an anomalous data point "outlier" is one that deviates by a certain standard deviation from the mean. Therefore, the simplest approach to identify irregularities is to calculate the mean, the first, second and third quartiles for

a selected group of turbines, and then, flag the data points that deviate from the range of the LIF (Lower inner fence) and UIF (Upper Inner fence) In Figure 9 there is the representation of these statistic concepts. Being represented as an outlier, all the points whose value is out of the range between the LIF and UIF .

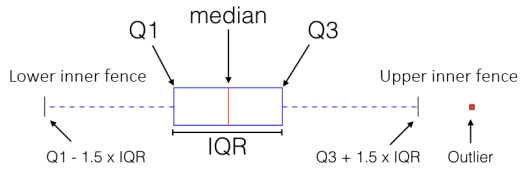

Figure 9: Box plot with fences

In order to do the analytic process over the wind turbines data, a Hive query was launched from a RScript in Spotfire.

One could think that this process could have been done in the Infrastructure module, but it was not possible since the anomaly detection process took dynamic parameters (park, turbine, date) corresponding with filters that the user could change from the Spotfire UI.

8.1.2 Visualization

The main objective of Visualization in Big Data, is to allow a comprehensible visual representation, which enables the users to glean insights from data, normally represented in dashboards. Previously to the visualization, the data was imported to Spotfire from the database, in this case, in order to improve the query latency Impala was used instead of Hive (see section 8.4.1).

8.2 Design

Given that the Exploitation module is divided into Analytics and Visualization, this section helps to understand this distribution representing its architecture diagram in Figure 10.

At the right side, there is Spotfire represented as a dashboard into the screen. This software has support of R language, which it is mainly used for managing tables. At the left part, there is the remote cluster, which represents the Hive and Impala services receiving connections from Spotfire. In the case of Impala, it is only used to import tables (see section Visualization). On the contrary of Hive that also does an analytic processes.

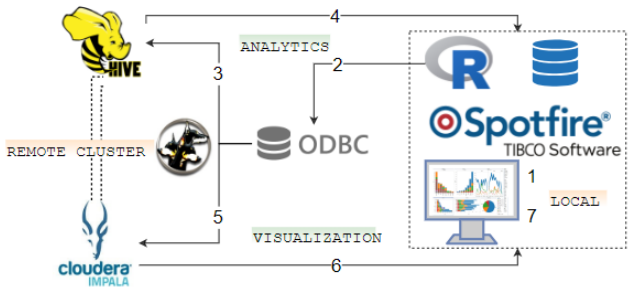

Figure 10: Extraction Design

8.3 Implementation

The different Analytics and Visualization steps are explained in this section, using as reference the numbers displayed in Figure 10.

8.3.1 Analytics

1. The analytic process starts when a user changes the analytics filters from the Spotfire UI, which it triggers a RScript that takes these parameters as arguments.

2. A connection to Hive in the remote cluster by its ODBC driver was created for each RScript launched. Moreover, as the cluster is protected using Kerberos to authenticate the user's computer, it was necessary to install and configure the MIT Kerberos(see 8.4.2).

3. This connection is used to launch a Hive job/query, which consists in calculate about the events of a day, the mean, the Q1 (first), Q2 (second or median) and Q3 (third quartile), the IQR or interquartile range (Q3 - Q1), the LIF (Q1 - 1.5*IQR) and the UIF (Q3 - 1.5*IRQ). To be able to do this calculations, the table was previously grouped by the day and signal type fields, also filtered using a WHERE clause with the parameters selected by the user (range of days, singnal type, aero and park). The statistics were calculated through the Hive function *PERCENTILE_APPROX(column, quartile)*[12]. Despite of the fact that Impala was much faster than Hive, it does not support this predefined function [13].

   Finally, the next sentence is the query launched from the RScript, changing the "park" and "signal type" arguments dynamically from the selected user parameters.

   odbcHiveConnection.sql("SELECT day, signal ,AVG(value) as AVERAGE ,percentile_approx(value, 0.25) - (1.5(percentile_approx(value, 0.75)-percentile_approx(value, 0.25))) as LIF, percentile_approx(value, 0.5) as Q2 ,percentile_approx(value, 0.75) + (1.5*(percentile_approx(value, 0.75)-percentile_approx(value, 0.25))) as UIF, unit FROM wtpoc.signals where aero IN ("aero3","aero5") AND signal IN ('signal12','signal30') GROUP BY day,signal,unit.*

4. This query returns an Output table which is represented in Spotfire as a graphic to visualize the anomalies, in which the mean, the median, the LIF and UIF are represented respectively by the medium red line, medium black line, lower line and upper line. Therefore, if the point value is lower than the LIF or higher than UIF, it is labeled as anomaly, represented by a red point (see Figure 11).

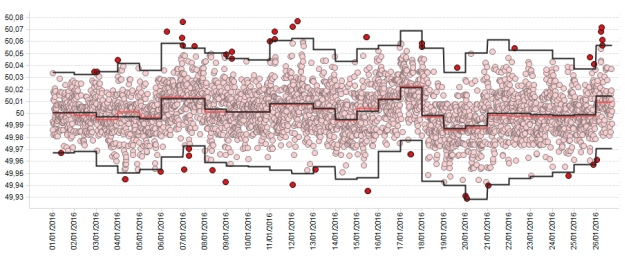

Figure 11: Outliers representation

8.3.2 Visualization

The visualization module mainly consists in, loading the data from Impala and representing them through the Spotfire UI.

5. The first process before the data visualization is to retrieve the data from the alarms, signals or analytics tables stored in the database. In this case Impala is used instead of Hive because it is faster (see section 8.4.1), using its ODBC driver to perform the connection between the user's computer with the Impala server in the remote cluster (also configured with the Kerberos authentication).

6. Therefore, having the alarms, signals and analytics tables loaded into the Spotfire system, it was time for the visualization. To do so, Spotfire has different representation forms, using those that provides the better insights and information to the final user.

7. The final result can be seen in Figure 12, in which different types of Graphics were chosen, including table representation, a plot with the sum of the time stopped, and a map that shows the different locations of each turbine (see section 7.3.2 Transform / Enrichment / Location).

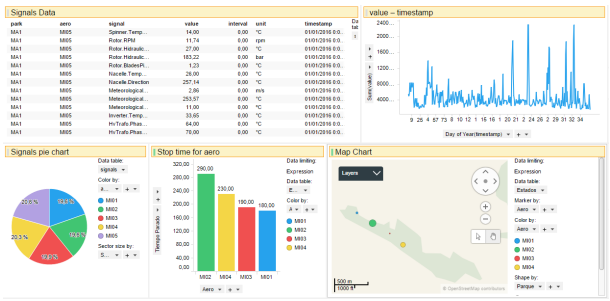

Figure 12: Dashboard example

8.4 Test

The Exploitation test is based on a comparison between the two data warehousing frameworks used, and explain how the Kerberos software was used to protect the data security.

8.4.1 Query latency

Before choosing the data warehouse framework to use to query the data in the Visualization module, the query latency of Impala and Hive was tested by making an SQL query over the signal's table. Table 7 is a comparison between the response times obtained from each tool. As it can be seen, Impala is much more faster than Hive in its queries because it uses its daemons running in all the nodes to execute a Massive Parallel Processing paradigm, instead of Hive that suffer latency in translating its SQL queries into Map Reduce jobs. Therefore, this is why Impala was chosen to load the data to Spotfire.

Table 7: QUERY LATENCY

Size	Hive	Impala
2.3GB	49,88s	0,30s
14,3GB	833s	5,96s

8.4.2 Security

There were not any problems with Kerberos when the jobs were launched using the shell through a SSH connection, but in the visualization the job had to be executed from a remote computer, enabling a connexion through an ODBC driver. Then, the MIT Kerberos software was installed and configured in the user's computer, and each day a new Kerbaros authentication ticket has to be created, using its corresponding keytab and principal.

9 RESULTS

In order to discuss the final results, the objectives defined at the beginning of the project must be checked and compared with the results obtained in the development.

The first objective of the PoC, the Ingest, was succefully accomplished. In this module the Hadoop shell was used to implement the data distribution from a the wind turbines sensors output files to HDFS.

The second objective, the Infrastructure module, was successfully achieved. This module consisted in the implementation of a Spark ETL process. In the Extract sub-objective, a Spark environment was set up and the data currently stored in HDFS was imported as Spark RDD. In the Transform sub-objective, the data was modeled through the Normalize, Formalize and Enrich processes. Finally, the Load was the last sub-objective, where the data from Spark was loaded into the Hive tables.

The third objective, the Exploitation, was successfully achieved. It was the last module of the data chain which contained two sub objectives. The first sub-objective, the Analytics, was successfully achieved through the implementation of an anomaly detection system over the wind turbines signals value by using simple statistical methods. The second sub-objective, the Visualization, which was achieved by representing the data in dashboards.

The different modules were integrated to create a single coherent system viewed as the data chain.

Last but not the least, the four objective was the application test which was achieved by delivering a quality software and the unit tests related with (scalability, error handling, heterogeneity, performance, security...) were done.

In conclusion, the PoC ended successfully on time, delivering to the client a new vision, concept and way to manage their wind turbines data.

10 CONCLUSIONS

Big Data & Analytics is a field that is becoming very popular in the industry sector, which every time, more companies are deciding to invest in its way to process and analyze their data. This project was required by a company of the energy sector which has a big amount of data (highly increasing) from sensors of its wind turbines, thermodynamic machines, solar panels, etc. Until now, managed through a traditional RBDM system. Then, the aim of the PoC was to introduce in the company a new way of handle their data, by developing a Big Data solution based on a Hadoop ecosystem.

As it was a Proof of Concept and for data privacy issues, they only provided us with one year data of a single wind turbine park.

The PoC was structured in the Ingest, Infrastructure and Exploitation modules, representing the different steps that the data pass through (chain of data).

The Ingest module was the first objective of the project which was necessary to establish the working environment. First of all, the data was migrated from the client computer to the Oesia cluster's MasterNode, then, these were distributed over all the cluster nodes using Hadoop.

The Infrastructure module was designed as a Spark ETL process. The Extraction process takes the data currently stored in the HDFS and imports it as Spark RDD, in order to be able to process these data in the Transformation process, which a different rules and processes (including Formalize, Normalize and Enrich) are applied to the extracted data in order to prepare it for the storage. Finally, in the Load module, the data are stored in the Hive database, being shared with Impala by the Hive metastore service.

The Exploitation was the last module of the data chain, and consisted in two processes. Firstly, the Analytics, that has the objective of discover anomalous behaviors through using simple statistical methods based on the values of the sensor's signals data, such as the mean, median, quartiles, interquartile range, lower inner fence and upper inner fence. Lastly, the Visualization was used to represent the data stored and analyzed through dashboards in Spotfire.

Finally, the test sections helped to analyze the impact in terms of time, used for the different tools or execution configurations. Anyway it always can be scaled by increasing the cluster resources.

In conclusion, the PoC ended successfully, giving to the client, a new vision, way and knowledge to how manage their wind turbines data, improving their latency time in generating graphics and views through doing faster queries to the database.

10.1 Future work

As future work, as it was a PoC which only used one year data of a single wind turbine park, the next step could be to take the whole data among all the different parks or sources, such as thermodynamical machines, solar panels, etc.

Moreover, as the system was designed to work using the Oesia's cluster, for its future production it should have to be migrated and adopted to the Client infrastructure.

In addition, the system could be redesigned to change from the current batch execution to a streaming one, which allows to see the wind turbine's information in real time.

This project also could be used as reference for Oesia, extrapolating it to other companies of the same sector.

11 Acknowledgements

This project would have not been possible without the support of Oesia Networks, which I thank Roser Avilés and Ruben Robles for their supervision and safety. Special thanks to my tutor, Ramon Martí and in one or another way to the Engineering School faculty of the Autonomous University of Barcelona. To finish, thanks to my family, without them, anything of this could have been possible.

References

[1] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[2] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60–68, 2012.

[3] M. Grover, T. Malaska, J. Seidman, and G. Shapira, *Hadoop Application Architectures: Designing Real-World Big Data Applications*. " O'Reilly Media, Inc.", 2015.

[4] "CDH 5.12.x Packaging and Tarball Information | 5.x | Cloudera Documentation," Jun 2018, [Online; accessed 21. Jun. 2018]. [Online]. Available: https://www.cloudera.com/documentation/enterprise/releasenotes/topics/tarball.html

[5] R. K. Wysocki, *Effective project management: traditional, agile, extreme*. John Wiley & Sons, 2011.

[6] G. PowerData, "Data lake: definición, conceptos clave y mejores prácticas," Apr 2018, [Online; accessed 21. Apr. 2018]. [Online]. Available: https://www.powerdata.es/data-lake

[7] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[8] "SparkContext (Spark 2.1.0 JavaDoc)," Dec 2016, [Online; accessed 24. Mar. 2018]. [Online]. Available: https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/SparkContext.html

[9] "GettingStarted - Apache Hive - Apache Software Foundation," Mar 2018, [Online; accessed 29. Mar. 2018]. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/GettingStarted

[10] "Spark SQL and DataFrames - Spark 2.3.0 Documentation," Feb 2018, [Online; accessed 25. Mar. 2018]. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html

[11] "7.1.6. What are outliers in the data?" Jan 2018, [Online; accessed 1. Jul. 2018]. [Online]. Available: https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm

[12] "LanguageManual UDF - Apache Hive - Apache Software Foundation," May 2018, [Online; accessed 19. May 2018]. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

[13] "Impala Analytic Functions | 5.14.x | Cloudera Documentation," May 2018, [Online; accessed 19. May 2018]. [Online]. Available: https://www.cloudera.com/documentation/enterprise/latest/topics/impala_analytic_functions.html