IT 15053
Examensarbete 30 hp
Juli 2015

Data Analysis on Hadoop - finding tools and applications for Big Data challenges

Juan De Dios Santos Rivera

Institutionen för informationsteknologi
Department of Information Technology

Abstract

Data Analysis on Hadoop - finding tools and applications for Big Data challenges

Juan De Dios Santos Rivera

With the increasing number of data generated each day, recent development in software, provide the tools needed to tackle the challenges of the so called Big Data era. This project introduces some of these platforms, in particular it focuses on platforms for data analysis and query tools that works alongside Hadoop. In the first part of this project, the Hadoop framework and its main components, MapReduce, YARN and HDFS are introduced. This is followed by giving an overview of seven platforms that are part of the Hadoop ecosystem. In this overview we exposed their key features, components, programming model and architecture. The following chapter introduced 12 parameters that are used to compare these platforms side by side and it ends with a summary and discussion where they are divided into several classes according to their usage, use cases and data environment. In the last part of this project, a web log analysis, belonging to one of Sweden's top newspapers, was done using Apache Spark, one of the platforms analyzed. The purpose of this analysis was to showcase some of the features of Spark while doing an exploratory data analysis.

Handledare: Markus Nilsson
Ämnesgranskare: Andreas Hellander
Examinator: Edith Ngai
IT 15053
Tryckt av: Reprocentralen ITC

Acknowledgements

I want to thank my supervisor Markus Nilsson from Granditude for giving me the chance of working this project, faculty reviewer Andreas Hellander from Uppsala University, the whole team at Granditude for being supportive and all of you who believed in me, in this work and have been in this adventure since day one. Thanks!

Contents

Chapter 1

Introduction

Each day more than 2.5 petabytes of data are generated. This data is generated from many different sources; data generated by automobiles, flight systems, online games, by every post someone does on a social network and even the items that a person bought at the supermarket are being stored somewhere for a purpose. The problem of this Big Data age is that while this number grows at an extraordinary rate, the physical space at our disposal is not increasing at the same rate.

Due to new computational challenges that arise from this situation and the need to solve the problem that comes with them, recent developments in open source software, provide the foundation needed to tackle these challenges. This group of software is highly scalable, distributed and fault-tolerant platforms created for the purpose of dealing with data in a different way for different ends.

The number of tools available in the market right now makes it somewhat difficult for someone to keep track of all of them, which is the reason why this project was born and the main motivation for doing it.

1.1 Motivation and goals

This project was born as a necessity to perform a study dedicated to select several open source platforms from the Hadoop ecosystem. The purpose of this research is to find appropriate tools and components that are suitable for the data mining challenges within the Big Data area. As mentioned before, because of the great number of tools in the market, getting to know and understanding each one of them can be challenging, thus we want to contribute to the community by giving a guide that will serve them at the time of making the decision of which platform to use and help them answering the following questions.

- Which tool is most suitable for a given task?
- What are the key features of this platform?
- What is the programming model for this platform?
- How does this platform compares against this other?

In general, our goal in this work is to present an overview of some of the tools available in the market for Big Data analysis. The overview will focus on the key features of the platform, its programming model and an insight of the architecture. Followed by comparing said platforms using certain parameters and classify them in a number of categories. The work will end with a practical application where we will select one of the tools from the study and perform a data analysis using said platform.

1.2 Method and structure

This work is divided in two main parts, a survey and a practical part. During the survey part, we will introduce the Hadoop platform and its principal components, followed by presenting the platforms that are the main focus of this work. The tools to be chosen need to comply with certain conditions: They need to be open source, they should be part of the Hadoop project or at least be compatible with it and the general purpose of the tools has to be either a data processing engine or query engine. At the beginning of the project a literature review was performed, looking for tools that complied with requirements mentioned. These are the seven chosen tools and their respective version used for this project.

- Apache Drill, version 0.7
- Apache Hive, version 1.1.0

- Apache Flink, version 0.9
- Apache Mahout, version 0.9
- Apache Pig, version 0.14.0
- Apache Solr, version 5.0
- Apache Spark, version 1.3.0

During the second part of the project, the practical section, a web log will be explored and analyzed using one of the tools mentioned in the previous list.

The following list shows the structure of the thesis and what will be done in each chapter.

- Chapter 2 introduces the Hadoop framework and its main components: HDFS, YARN and MapReduce.
- Chapter 3 introduces the tools mentioned above and give an overview about their key features, how it works under-the-hood and their programming model.
- In chapter 4 the tools will be further analyzed. First, they will be compared according to certain parameters, followed by a summary where they will be categorized.
- Chapter 5 is about the data analysis done in the project. This chapter outlines the problem, methods used in the analysis, the results and a discussion about the analysis and about the platform chosen performed doing the job.
- Chapter 6 presents a discussion of the results obtained in chapter 5, suggestions for future work and a section stating the limitations encountered during the project.

1.3 Related work

Many of the papers about this topic written before, usually focus on a single platform, in comparison between two of them and analysis about their performance. One of the most referenced articles in this work and one that is very similar, is about Apache Drill. The article compares Drill against other query platforms such as Apache Hive while exposing its main features and the way it works [1]. One of the first papers about Apache Spark, written by Zaharia et al., introduces the main abstraction for programming in Spark as well as the main components of the platform [2]. Gates et at., in their paper written at Yahoo!, give an system overview of Pig, an introduction to its programming language and benchmark results regarding the performance [3]. Apache Flink is briefly described in a paper written by Markl [4] where the author also writes about the Big Data era.

Chapter 2

Hadoop framework

Apache Hadoop [5] is an open source and distributed framework run across a cluster of commodity hardware used for the processing, management and storage of large data sets. The framework, which is able to scale up from one single machine up to several thousands and able to handle petabytes of data, is fault tolerant, which means that it was designed to handle failures.

Besides the *Hadoop Common* module, which contains the main utilities needed by Hadoop, the framework includes three additional modules:

- **Hadoop Distributed File System (HDFS):** Fault-tolerant distributed file system for storing data across commodity machines.
- **Hadoop YARN:** Architectural center of Hadoop that manages the cluster.
- **MapReduce:** Programming model for the processing of large data sets.

2.1 Hadoop Distributed File System

The Hadoop Distributed File System [5, p. 41] or HDFS is a highly scalable and fault-tolerant distributed file system for Hadoop, designed to run on commodity machines.

An HDFS cluster adopts a master/slave architecture that consists of one *NameNode* and *DataNodes*. The NameNode, which takes the role of the master, is a server responsible of managing the cluster, its metadata and the DataNodes (slaves) and it performs file system operations such as opening, closing and renaming of the files and directories on the file system.

The DataNodes is where the data is stored. The content of a file in HDFS is divided into several blocks that are stored into numerous DataNodes (this mapping is done by the NameNode). To maintain fault-tolerance, reliability and integrity of the data, these blocks are also replicated and monitored by the NameNode. In the case of a failure where any block is lost, the NameNode, creates another one. Figure 2.1 illustrates this.

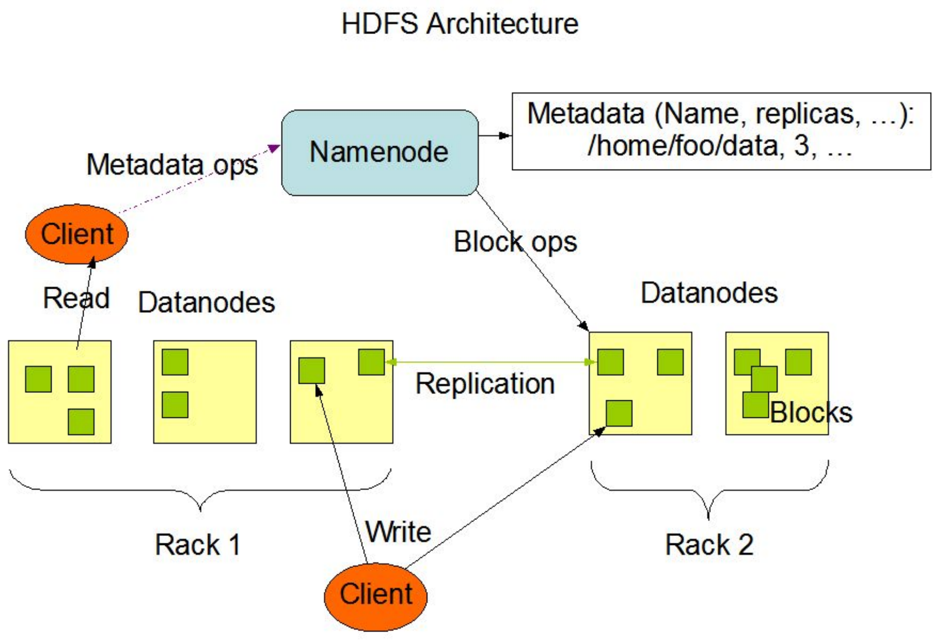


Figure 2.1: HDFS architecture. The light blue square represent the *NameNode*, the responsible of managing the cluster. The yellow squares are the *DataNodes*, where the data is stored (Image from [6]).

HDFS also has some features such as, *rack awareness* [7], which takes into consideration the physical location of the node at the time of using it, a *balancer* whose purpose is to balance the disk usage of the cluster across the NameNodes, a utility to keep track of the health of the system, named *fsck* and the ability to rollback after performing an update.

2.2 Hadoop YARN

YARN (Yet Another Resource Negotiator)[8], also known as MRv2 (MapReduce 2.0), is the architectural center in charge of managing the multiple processes run by Hadoop. It was built out of a necessity to tackle the bottleneck created by the previous architecture, called MRv1 or MapReduce v1. In the previous architecture, a service called *JobTracker* had two primary responsibilities, resource management and job scheduling or monitoring, a daemon that creates and assigns tasks to all the node of the cluster. This previous architecture caused a bottleneck if the cluster consists of a large amount of nodes.

Under YARN, the responsibilities of the JobTracker, are separated into different daemons: *ResourceManager*, *NodeManager* and *ApplicationMaster*.

The ResourceManager is a global daemon (one per cluster) whose purpose is the management of the resources across all the applications running on the system. The NodeManager is a per-node slave, that supervises the *Container* (resource) of the application and is responsible of said container and monitoring the elements of it, such as CPU, disk and memory.

Lastly, the ApplicationMaster is a per-application daemon that inquiries the resources from the ResourceManager. Figure 2.2 shows the YARN architecture.

2.3 MapReduce

MapReduce [10][11] is a framework and programming model for writing applications (jobs) that process a big amount of data in parallel among a cluster. A MapReduce job consists on two major parts: the map task and reduce task.

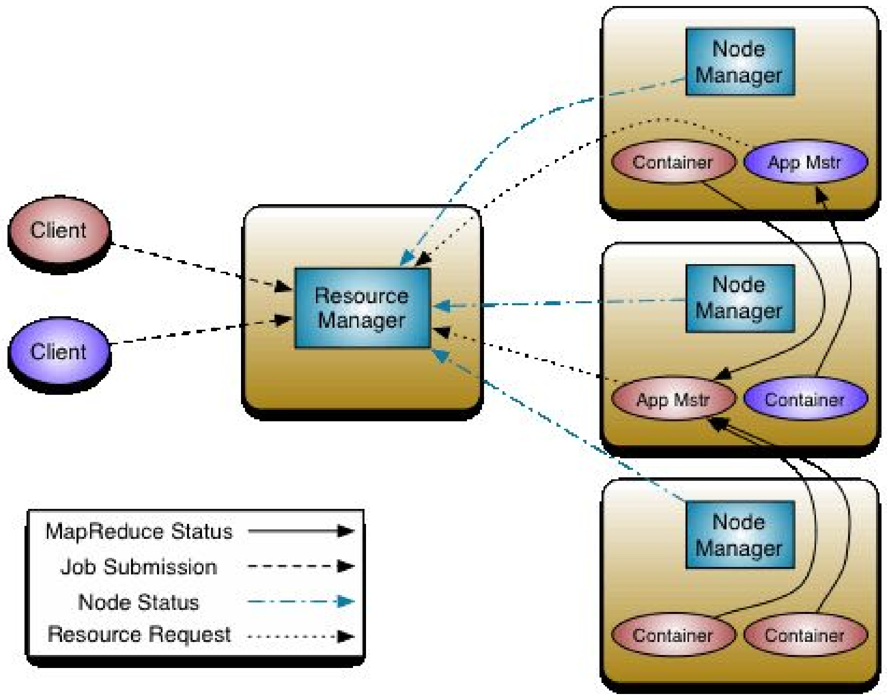


Figure 2.2: YARN architecture. The *ResourceManager* (one per cluster) manages and schedule the jobs of the cluster. The multiple *NodeManager's* supervises the resources of the node and inquiries the needed resources from the ResourceManager (Image from [9]).

During the map part, a function *Map*, splits the input (usually stored in HDFS) into several chunks that are viewed a as `<key, value>` pair. The output of the Map function is another set of `<key, value>` pairs that are sent to a combiner, whose job is aggregate this output after being sorted on the keys.

The *Reduce* function, receives as input the output of the combiner which is another `<key, value>` pair and it is the responsible on solving the actual problem that was tasked by the master node. It does it by combining the results from the input, typically using an aggregating function. The output is sent to the HDFS. Figure 2.3 shows the MapReduce process.

2.3.1 MapReduce example

Suppose we are running a simple job that counts the occurrence of each word from a file. After reading a sample input, the first map emits:

```
<Hello, 1>
<World, 1>
<Bye, 1>
<World, 1>
```

Then a second map function, emits:

```
<Hello, 1>
<Hadoop, 1>
<Goodbye, 1>
<Hadoop, 1>
```

Each of these maps, we will sent to the combiner separately, producing the following result for the first map:

```
<Bye, 1>
<Hello, 1>
<World, 2>
```

And this for the second map:

```
<Goodbye, 1>
<Hadoop, 2>
<Hello, 1>
```

These outputs processed by the Reduce function which will sum up the values of the occurrences, producing the following and final output:

```
<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 2>
```

This example was taken from the official MapReduce website [10].

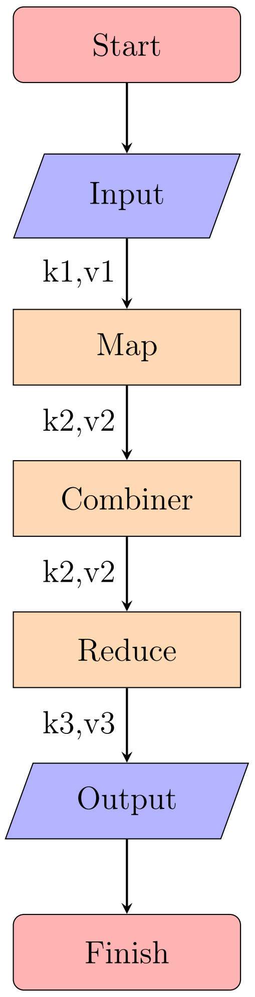


Figure 2.3: Execution phases of a MapReduce job. During the Map stage, a function is applied to the split input, then the emitted output from Map, goes to a local combiner. In the Reduce stage, the results of the Map function are combined.

Chapter 3

Hadoop-related projects and ecosystem

Throughout the years, Hadoop has been extended to much more than a framework for performing MapReduce. Features such as HDFS, the reliability, scalability and the architectural center YARN, makes Hadoop a platform to develop for. Since its creation, the Hadoop ecosystem has being extended to much more than the components previously mentioned. Now there are a number of engines, libraries and frameworks that have improved the system and extended it. The functionalities of the tools extends from new ways to access, interact and analyze the data (batch and real-time), new ways of programming data applications, database and query systems (SQL and NoSQL), tools to improve the security, tools to view the data and more.

For this project we are focusing on tools whose function is to analyze data, mining data or perform queries over the data set.

Throughout this chapter, we will introduce several platforms from the Hadoop ecosystem and present an overview of them that will cover their main features, programming model, architecture and components. These are platforms we found suitable for these tasks mentioned in the last paragraph and tools we will be analyzing and studying in the upcoming parts of the project.

3.1 Apache Drill

Apache Drill[12] is an SQL query engine for distributed file systems and NoSQL databases, inspired by Google Dremel[13], a distributed system for performing queries over big data sets. The main idea of Drill, is to offer a low-latency, flexible and familiar SQL-based query system.

3.1.1 Key features

- Drill offers a dynamic and schema-free JSON model for performing queries.
- Drill is extensible to several data sources. The data can be obtain from sources such as the local file system, HDFS, Apache HBase [14], Amazon Simple Storage Service (Amazon S3) [15], and NoSQL databases like MongoDB [16].
- Drill provides a Java API for implementing user defined functions (UDF).
- Provides an UI platform called Drill Web UI for monitoring and canceling queries.
- Drill features a Plug and Play integration with Apache Hive that allows it to perform queries in Hive tables, make use of already designed Hive UDFs and supports Hive file format.
- Drill supports a familiar and standard revision of the SQL language, SQL:2003.

3.1.2 Programming model

Drill is a schema-free SQL query engine where the data is internally represented as a JSON-like data model. Being schema-free means that Drill does not need to know the schema or definition of the data because it is discovered dynamically as the data comes in. Additional to this, it offers *in situ* access to the data, meaning that the data can be extracted directly from the source [1].

3.1.3 How it works

The main service of Apache Drill, Drillbit, is responsible of accepting the jobs of the client, as well as launch and process the queries and return the results. While running in a cluster, Drill needs Apache Zookeeper [17], a platform created for the purpose of configuration and synchronization of distributed systems. Apache Drill does not follow a master-slave architecture, therefore any Drillbit from the cluster can accept a request and the Drillbit that gets it, become the "organizing" Drillbit for the duration of the task[18].

The typical flow of a Drill query goes like this: Once the client issues a query and it gets accepted by a Drillbit (that will be become the driving Drillbit), it parses the query and uses Zookeeper to find the other Drillbits from the cluster and then, it determines what nodes are appropriate for performing the job. After each node finishes its part, they return the data to the driving Drillbit who then returns the result to the client.

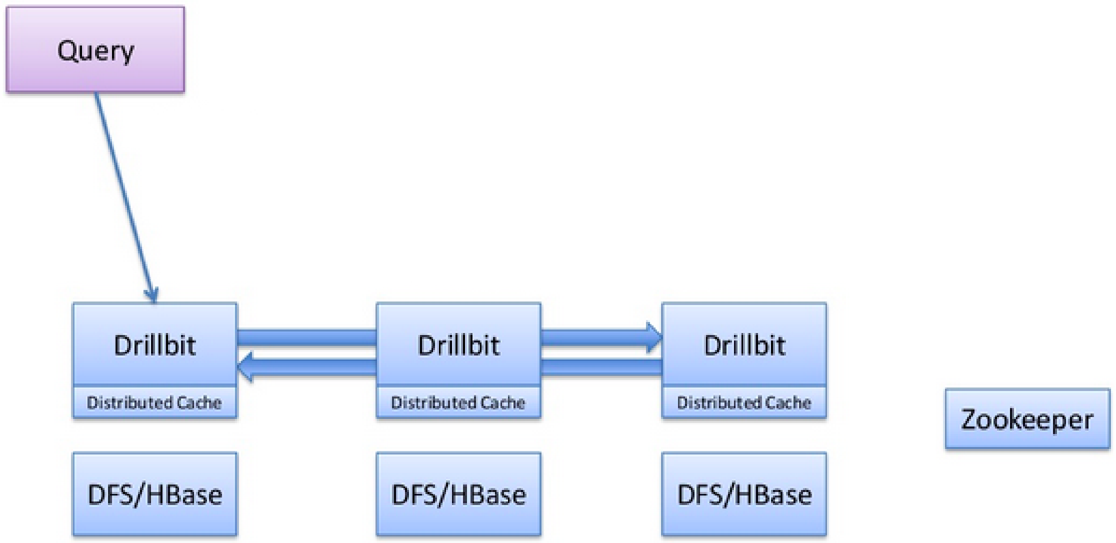


Figure 3.1: Drill's basic query flow. At first, the user submit a query which is accepted by a Drillbit whose function is to determine which are the appropriate nodes for performing the job (Image from [18]).

3.2 Apache Flink

Apache Flink [19], also known as Stratosphere [4] before being part of the Apache Software Foundation, is an open source engine for distributed, large-scale data processing. By being an engine, Flink is capable of running in standalone mode, yet it can also run on top of Hadoop's HDFS and YARN.

3.2.1 Key features

- Flink is optimized for cyclic and iterative computations by making use optimized collection-based transformations.
- It supports both in-memory batch processing and data streaming analysis via Flink Streaming API.
- Automatic optimization allows it to run on single computers up to clusters of hundreds of machines.
- It has a vertex-centric graph computation API, called Spargel.
- It manages its own memory and performs the caching and processing of data in a dedicated memory fragment.
- Programs can be written in Java and Scala.
- Compatibles with platforms such as Hadoop, HBase, Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

3.2.2 Flink high level tools

As mentioned in the previous section, Flink features two high level components, Flink Streaming and Spargel for data stream analysis and graph processing respectively. Sect performing different tasks (Flink Streaming and Spargel). Section 3.2.2 and 3.2.2 introduces them.

Flink Streaming

Flink's batch processing counterpart, Flink Streaming is an extension of Flink's core API made for the processing and analysis of streams of data. The basic abstraction of Flink Streaming is an object called `DataStream`. This object represents stream of data from either a data source such as Flume or Twitter (more on section 4.1.3) or an already transformed data stream. The API introduces various operations for performing transformations on the data stream that are grouped into three categories: Basic operations, grouped operations and aggregations. These operations work over individual data points or over windows (chunks) of data. After the data is processed, it is output to either the standard output, as plain text file or as CSV.

Spargel

Spargel is Flink's API for graph processing. A graph processing job in Spargel is made of sequences of supersteps that consists on two operations, sending messages to other vertices and receive messages from other vertices to update a vertex own state. Because it relies mostly on operations performed over vertices, it is known as a vertex-centric platform. Figure 3.2 shows an example of a Spargel application that propagates the minimum vertex ID across the graph. At the initial step, each of the vertex, which has a key and a value, contain an initial value. On the first superstep, vertex 2 is updated with the value 1, since it is the smallest ID from all of its neighbor. At the second superstep, this value 1 is propagated to vertex 3 and 4 since they are connected to vertex 2.

3.2.3 Programming model

Flink's programming model is based on transformations, such as aggregate, cross, filter, iterate, join and union done on distributed data sets. The input could be from an data set or a collection of items created in the program, e.g a list of strings, and the output is written either to one or several files or as an standard output in the console.

Example Suppose we have on a database a table `EMPLOYEE` with the columns `ID` and `NAME` and a serialized log file stored in a file system that contains the position between a company of an employee. First, we perform a query on the database to get the table and store in a data structure, then we read the log file, deserialize it and store it in another data structure. Then we perform the join operation like this:

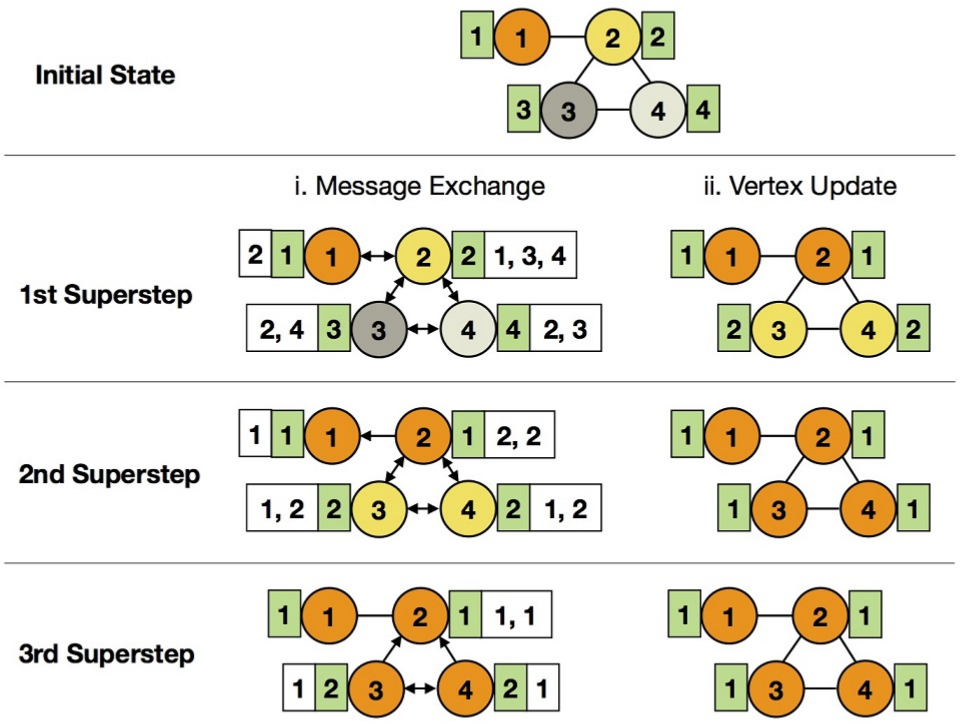


Figure 3.2: Spargel example. This image shows how the vertices are updated at each superstep (Image from [20]).

```
DataSet<Tuple2<Employee, Position>> employeePosition =
employeeList.join(positionList).where("ID").equalTo("userID");
```

Resulting in a tuple of all the employees and their positions among the company. Idea of the example taken from Flink's official website [19].

3.2.4 How it works

Flink is made of two main processes: the *JobManager* and *TaskManager*. The JobManager functions as a coordinator, it has the responsibility of scheduling, managing the resources of the system and deployment of the tasks. The TaskManager is the one who executes the job and it is constantly sending its status to the JobManager. In addition to these processes, there is also the *client* that is created when a job is submitted and is the responsible of compiling said program, optimize it and to send it to the JobManager. Figure 3.3 shows this process.

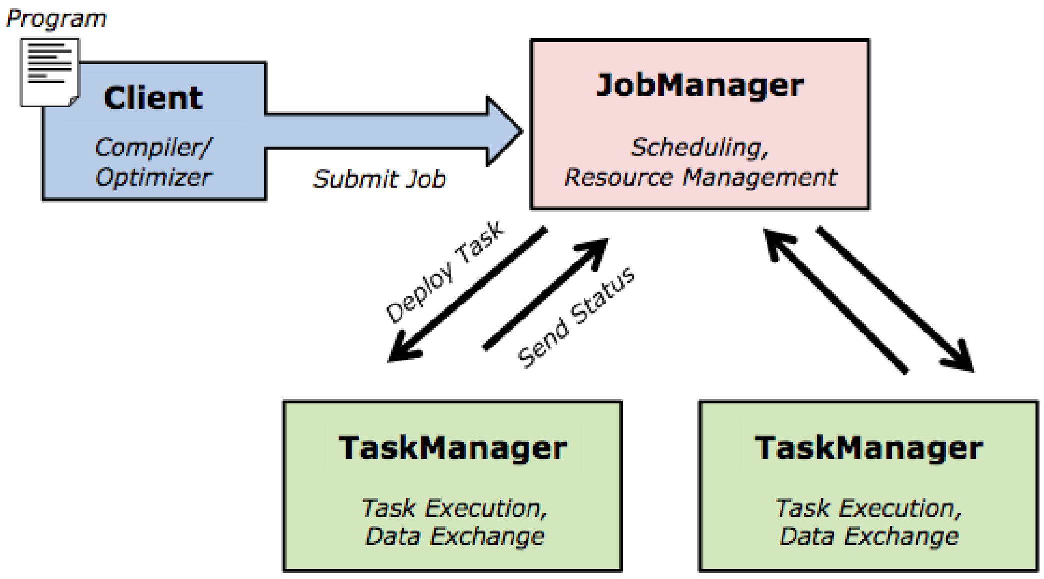


Figure 3.3: How Flink works. After the client submit a job, it is coordinated and managed by the *JobManager*, who sends it to the *TaskManager*, the responsible of executing the job (Image from [21]).

3.3 Apache Hive

Apache Hive [22] is a data warehouse tool for analysis, exploring and querying of data on the Hadoop ecosystem that provides tools for data extract/transform/load (ETL). Created with the purpose of simplifying writing MapReduce programs, Hive's scalability, extensibility and ease to use, has made Hive one of the de-facto tools for Hadoop.

3.3.1 Key features

- HiveQL, the query language for Hive, is based on SQL and it can be extended using UDF, user defined types (UDT) and custom MapReduce scripts [23].

- Compatible with file systems such as Amazon S3, HBase and HDFS.
- When used on top of Apache Tez, a framework for performing directed-acyclic-graph of tasks for processing data, the performance of Hive increase since it reduces the number of map/reduce jobs executed [24].
- A web interface named *HiveWebInterface* allow the user to browse through the metadata and schema of the data, launch queries and check their status.
- It support indexes for faster queries.
- Supports several file types such as text file and Hadoop's SequenceFile. It also operates on compressed data.
- Hive includes HCatalog, a table management system for Hadoop that allows other processing tools of the ecosystem to more easily access the data.
- Hive includes a server named HiveServer, built on Apache Thrift that allows a client to connect remotely and submit requests using several programming languages. In addition, it also supports ODBC (Open Database Connectivity) and JDBC(Java Database Connectivity).

3.3.2 Programming model

Hive employs an SQL-like language, called Hive Query Language (HiveQL) for querying data stored on the Hadoop cluster. HiveQL provides an easier way to run MapReduce programs and to explore the data because when an HiveQL query is executed, it is translated into a MapReduce job. Due to its nature and the design of Hadoop, Hive does have certain limitations, for example, the queries suffer from latency issues due to the overhead created by MapReduce, making it a more suitable tool for data warehousing, where the data stored is usually static.

3.3.3 How it works

The design of Hive is made of five main components: UI, driver, compiler, metastore and execution engine.

The UI, which could be a command line interface or a web-based GUI, is where the user executes the query, which is sent to the driver. At the driver, a session is created for the query and it is sent to the compiler. The compiler parses the query, gets all the information (metadata) needed about the query from the metastore and generates an execution plan, which is DAG of stages where the each stage could be either a map or reduce job. The last component, the execution engine, executes the plan and provides an output that is stored in the HDFS.

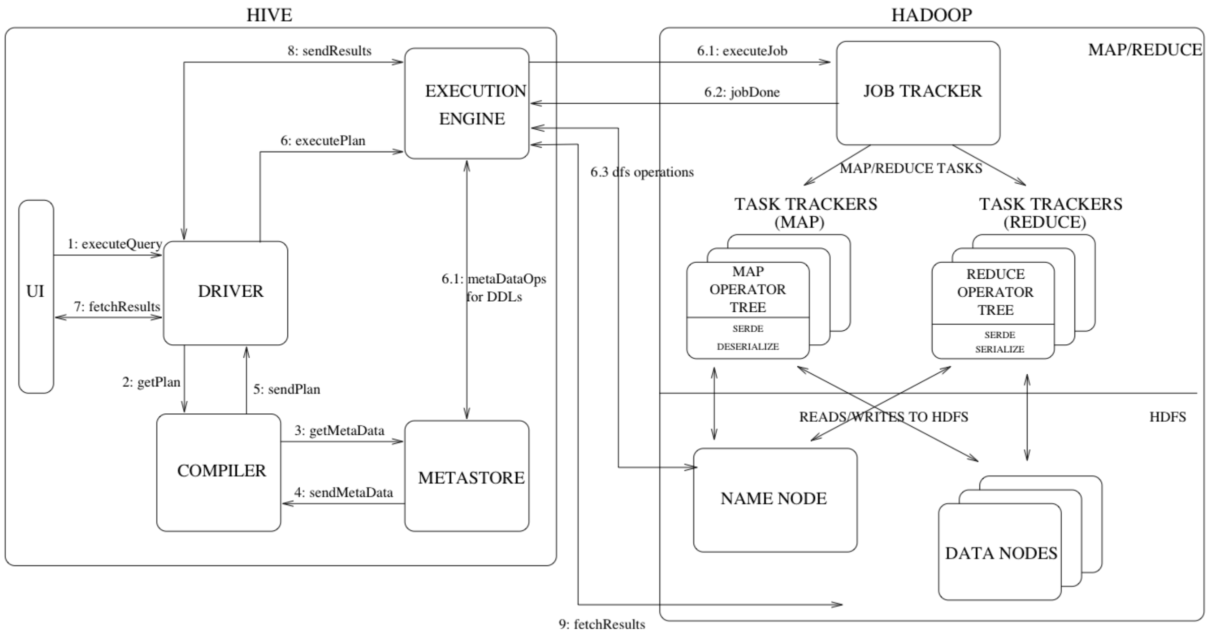


Figure 3.4: Hive's architecture. The UI is where the user enters the query, the driver creates a session for the query, the compiler parses the query and generates the execution plan, the metastore is for extracting the metadata and the execution engine, executes the plan (Image from [25]).

3.4 Apache Mahout

Apache Mahout [26] is a library designed to perform scalable machine learning algorithms. Machine learning is a discipline with strong ties to fields such as artificial intelligence and statistics, that instead of following explicit programmed instructions, it uses data to teach the machine how to build a model that will learn from previous outcomes. By being a library, Mahout can be used in a regular Java project and other platforms such as Apache Spark, Flink and in a MapReduce job. While this may be flexible for the developer since it does not lock him into using a particular platform, not every algorithm is compatible with every platform, some of them are made to be executed in local mode, distributed mode (For Hadoop) and for Spark.

3.4.1 Algorithms

Mahout algorithms support three main cases:

- Classification: A classification algorithm learns about the existing categories of a data set through training data and then uses the model and what it learned to classify unknown cases. Classification is part of a subfield of machine learning, called supervised learning.
- Clustering: Task of grouping observations from a data set based on their properties. Clustering is part of a subfield of machine learning, called unsupervised learning.
- Collaborative filtering: Techniques used to learn about the user preference through mining of its behavior. This is mostly used in the recommendation systems used in platforms like Netflix[27].

Besides these algorithm, Mahout also introduces others algorithms for tasks like dimensionality reduction and topic model.

Table 3.1 presents some of the algorithms from Mahout that can be executed using Hadoop.

3.5 Apache Pig

Apache Pig [28] is an ETL platform for analysis and processing of large data sets on Hadoop. It introduces a high-level scripting language, Pig Latin [29] for performing MapReduce jobs using a procedural approach.

Algorithms in Apache Mahout		
Algorithm	Case	Description
Naive Bayes	Classification	Probabilistic classifier based on Bayes' Theorem
Random Forest	Classification	Learning method for classification through the use of multiple decision trees
K-means clustering	Clustering	Divides the data set into k different partitions followed by assigning each observation to the cluster with the nearest mean
Fuzzy k-means	Clustering	Similar to regular k-means with the exception that one observation could belong to more than one cluster
Streaming k-means	Clustering	Similar to regular k-means with the exception that it introduces features to deal with data streams
Spectral clustering	Clustering	Clusters the observations by using it spectrum based on the similarity matrix of the data set.
Item-based collaborative filtering	Collaborative filtering	Single item-based recommendation by using previous cases of prefered similar items.
Matrix factorization with Alternating Least Squares	Collaborative filtering	Recommendation of several items based on the user preferences by items.

Table 3.1: Algorithms available in Mahout.

3.5.1 Key features

- Pig supports several execution modes. Besides running on local mode or in MapReduce mode using Hadoop and HDFS, Pig can also run on top of Apache Tez, which is an engine built on top of YARN for building data processing applications.
- Can be run on either interactive mode [30, p. 19], where the user executes the commands and statements on a command line interface or in batch mode using a script.
- Pig Latin supports UDF written in Java, Python and more.
- Pig is self-optimizing. This way the user does not have to focus on optimization of the scripts.
- Can execute multiple queries at once.

3.5.2 Programming model

Pig Latin is a parallel dataflow programming language, where "*a user specifies a sequence of steps where each step specifies only a single, high-level data transformation*" [29, p. 1100]. Unlike a query language where you ask a question about the data without specifying the how to, using Pig Latin the user knows exactly how the data will be transformed (not necessary in the same order they were written).

Pig Latin data model consists of four types: atoms, tuples, bags and maps, with the last three also known as relations. The atom, the simplest of them, is an atomic value like a number or string. A tuple, is an ordered collection of fields and a collection of tuple is called a bag. Lastly, the map is a collection of items that are associated with a key, think of it as a hash map of `<key,value>` pairs.

The elemental construct used in Pig Latin to process data is called a statement. The statement is an operator that takes one of the previously mentioned relations as an input, an expression or schema and produces another relation.

Pig Latin also support control structures (via UDF) and has several built in functions, such as `AVG` and `COUNT`.

3.5.3 How it works

The execution plan of a Pig program, goes through five steps. Upon execution, it first goes to the parser where the syntax, type and schema is checked and it outputs a logical plan arranged in a directed acyclic graph (DAG). The logical plan goes to the logical optimizer, where optimizations are performed and this new optimized logic plan gets compiled into a series of map and reduce jobs, by the map/reduce compiler. Once again, it goes through another optimizer (map/reduce optimizer) and its result is a DAG of optimized map-reduce jobs topologically sorted that are emitted to Hadoop for execution [3]. Figure 3.5 shows this process.

3.6 Apache Solr

Apache Solr [31] is an open source platform that focuses on the search and indexing of information and resources within an enterprise; this is known as a enterprise search platform [32]. Solr is part of the Apache Lucene project [33], an information retrieval software library.

3.6.1 Key features

- Solr supports writing and reading from HDFS.
- One of the main features of Solr is its ability for full-text searching and its near real-time indexing.
- Features within the search process include faceting, an arrangement of search results into groups, clustering of the results into groups based on similarities and *MoreLikeThis*, a feature that allows running new queries focusing terms returned by a previous query.
- Introduces a web client, called Solar Admin UI that offers several functions such as running searches, view the log files of the system, management of the cores, access online documentation among others.

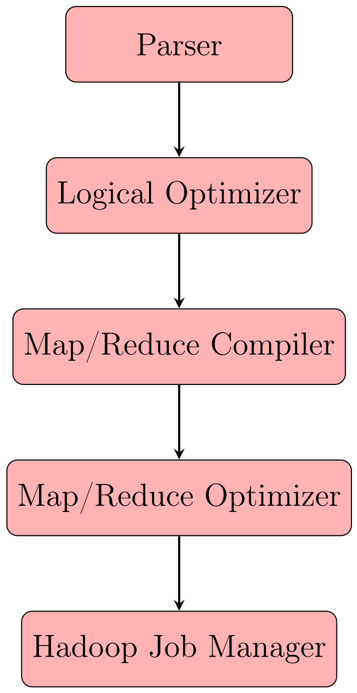


Figure 3.5: Pig execution phases. At the first stage, parsing, a logical plan is produced and sent to the logical optimizer. At the logical optimizer, a new logic plan is produced and sent to the Map/Reduce compiler, followed by the Map/Reduce optimizer. Lastly, the map/reduce jobs are sent to the job manager.

- Supports server statistics via Java Management Extensions (JMX) which is a tool for managing and monitoring applications.
- Solr can be queried over many interfaces like REST clients, wget and native clients made for various programming languages such as Python and C#.
- Through Zookeeper, the cluster is automatically managed and coordinated thus the user does not have to worry about scaling up or down.

3.6.2 Overview of searching

To start searching in Solr, the first step is to launch the query from any of the sources mentioned above. In Solr, a user is able to search for a single term, a phrase or a combination of them. While writing the search query, the user is able to specify the format of the response which can be in JSON, XML, formats that are extensions of others to make the response safe to be interpreted by several programming languages and others. Optional to this, it can be specified in the query if the response should contain any faceting. This is an example of how to execute a search using cURL to find the occurence of the phrase "hello world" and getting the response in JSON format.

```
curl "http://localhost:8983/solr/directory/select?
         wt=json&indent=true&q=\"hello+world\""
```

On a lower level, when a query is executed it is processed by a request handler who defines the logic of the request followed by the query parser. At the parser, the query, parameters and filters specified by the user are interpreted. Figure 3.6 shows this process.

3.7 Apache Spark

Apache Spark [35] is an engine for distributed large-scale data processing characterized by its ability to perform computations in memory, its multiple APIs and components.

3.7.1 Key features

- Instead of disk-based operations, Spark loads the data into the cluster memory and performs in-memory operations using a directed acyclic graph (DAG) engine.
- Can run on top of different cluster management platforms such as Hadoop YARN and Apache Mesos or in standalone mode.
- Data sources include Amazon S3, HDFS, HBase and the local file system.
- Supports APIs for writing applications in Java, Python and Scala.
- Includes four components for performing tasks such as graph processing (GraphX), queries (Spark SQL), streaming analytics (Spark Streaming) and complex analytics via a machine learning library (MLlib).

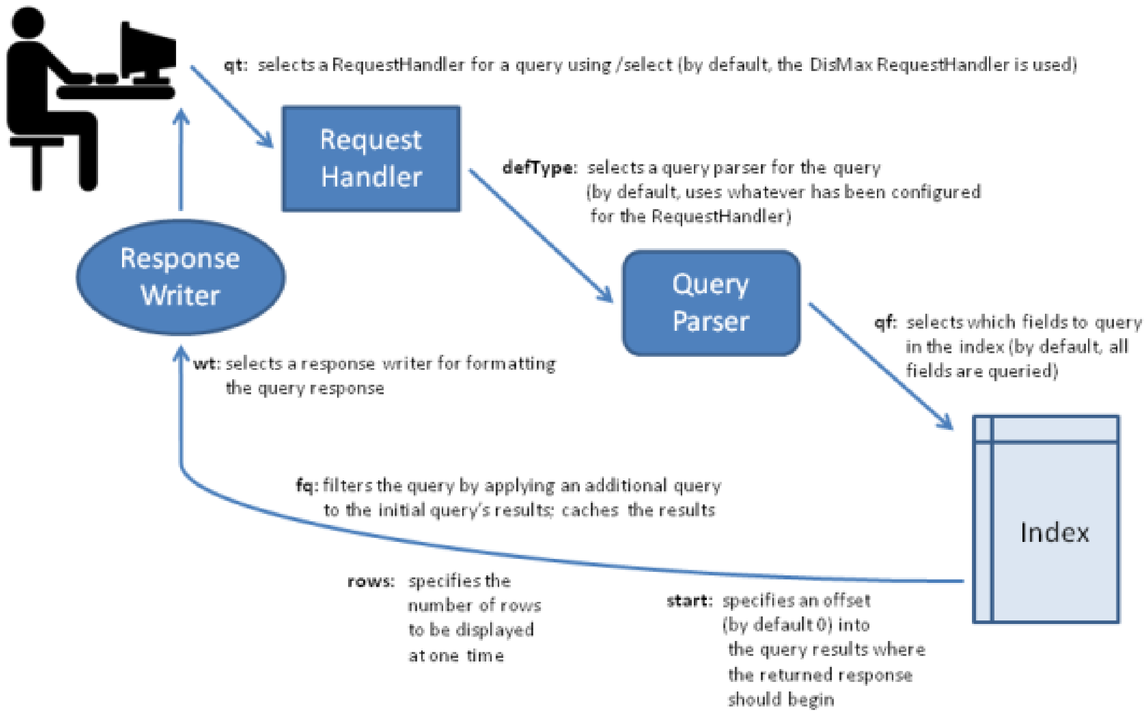

qt: selects a RequestHandler for a query using /select (by default, the DisMax RequestHandler is used)
Request Handler
defType: selects a query parser for the query (by default, uses whatever has been configured for the RequestHandler)
Query Parser
qf: selects which fields to query in the index (by default, all fields are queried)
Index
Response Writer
wt: selects a response writer for formatting the query response
fq: filters the query by applying an additional query to the initial query's results; caches the results
rows: specifies the number of rows to be displayed at one time
start: specifies an offset (by default 0) into the query results where the returned response should begin

Figure 3.6: How Solr works. When a query is executed it is processed by a request handler. Then, at the query parser, the parameters and filters specified by the user are interpreted (Image from [34]).

- Offers a web-based interface that display information about the running applications and about the cluster (while in standalone mode).
- Spark introduces an abstraction called Resilient Distributed Dataset (RDD), which a distributed collection of items.

3.7.2 Programming model

The programming model of Spark is mostly based in two main abstractions: RDD and shared variables (broadcast variables and accumulators).

An RDD or Resilent Distributed Dataset, Spark's primary abstraction, is a collection of items distributed across Spark's nodes that is manipulated in parallel. It gets its data either from an existing collection in the program, also known as driver program, from an external source or by transforming other RDDs.

The second abstraction of Spark is shared variables. During the execution of a program, Spark makes copies of all the variables in each of the nodes and the program works on separated copies of them, which might result in unnecessary overhead. To avoid this, Spark provides two types of shared variables: broadcast variables and accumulators. A broadcast variable is a read-only variable that is cached on each machine, instead than having many copies of it in all the instances of the running application and an accumulator is a variable whose value can only be added(e.g. a counter).

3.7.3 Spark high level tools

Spark features four high level tools for performing different tasks. In this section, we will give an overview of them.

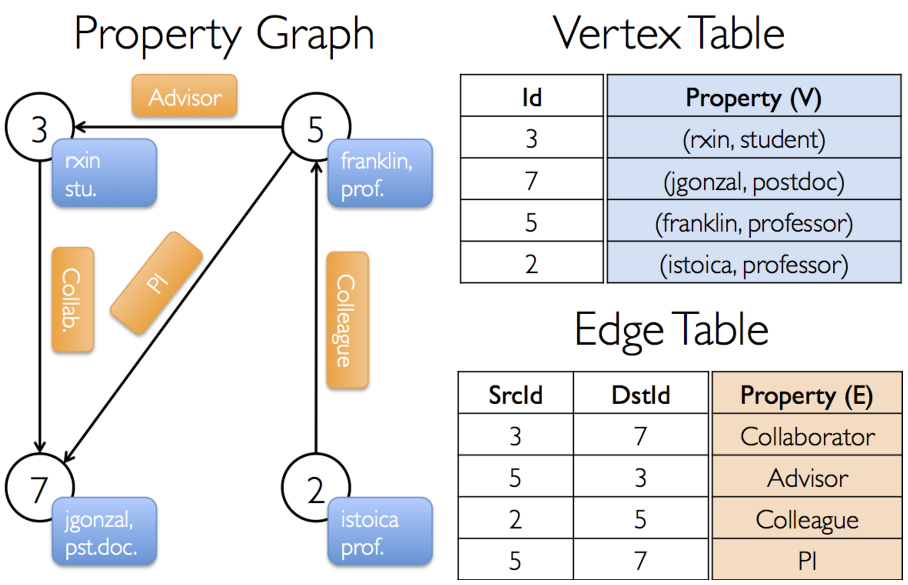
GraphX

GraphX [36] is Spark's graph processing API with the goal of unifying the data and graph computations under one same API without the need of moving between the graph and the tables containing the data. GraphX introduces an object called Resilient Distributed Property Graph, a *"directed multigraph with user defined objects attached to each vertex and edge"* [37], meaning that two different nodes can have multiple parallel edges between them. Property Graph is an extension of Spark's RDD thus the graph can also be seen as a collection. Figure 3.7 shows an example of Spark's Property Graph.

GraphX includes several operators and algorithms such as: number of edges, number of vertices, reverse graph, join vertices, PageRank, connected components and triangle count.

Spark SQL

With Spark SQL [38, p. 161], Spark is able to perform queries expressed in SQL or HiveQL within a Spark program, using any of the supported pro-



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Figure 3.7: Example of Spark Property Graph. This image shows custom user defined objects attached to the vertices and edges of the graph (Image from [37]).

gramming languages APIs mentioned above. Spark SQL presents an object called DataFrame, a table-like abstraction made of rows, named columns and a schema that describes the data, in other words, it is a representation of a relational database table. The schema of the DataFrame can be inferred in some cases, such as when working with JSON datasets, otherwise, it has to be specified by the user.

Like Apache Drill, Spark SQL also supports data stored in Hive tables, queries and UDFs.

Spark Streaming

The core Spark API has an extension named Spark Streaming [39] that introduces a fault-tolerant, streams processing data platform that uses the same programming API used for batch jobs, meaning that the jobs can be written in a similar way to those for batch processing. Said streams of data comes from sources such as HDFS, Apache Flume, Apache Kafka and or

TCP.

A continuous stream of data in Spark Streaming is represented by an abstraction called *DStream*, which is a sequence of the RDD abstract object mentioned before. As the data is arriving, it is divided into batches, followed by sending each of these batches to the engine, resulting in a final stream of results that are sent to a dashboard, database or file system.

MLlib

The last of these tools is a machine learning library named MLlib [38, p. 213] that support the following cases:

- Basic statistics: summary statistics, correlations, stratified sampling, hypothesis testing and random data generation
- Classification and regression: linear models, naive Bayes and decision trees (Random Forest and Gradient-Boosted Trees)
- Collaborative filtering: alternating least squares
- Clustering: k-means
- Dimensionality reduction: singular value decomposition and principal component analysis
- Feature extraction and transformation: *TF-IDF* and *Word2Vec*

3.7.4 How it works

At a high level, a Spark cluster is made of three components: a coordinator named *SparkContext*, a cluster manager and the worker node. SparkContext lies in the main program (or driver program) and it is responsible of making the connection between the program and the cluster manager, scheduling the resources within the applications and sending the tasks to the worker node. The cluster manager, which could be the standalone manager or one such as YARN, schedule the resources across the applications. Lastly, there are the worker nodes consisting of an executor that runs the tasks.

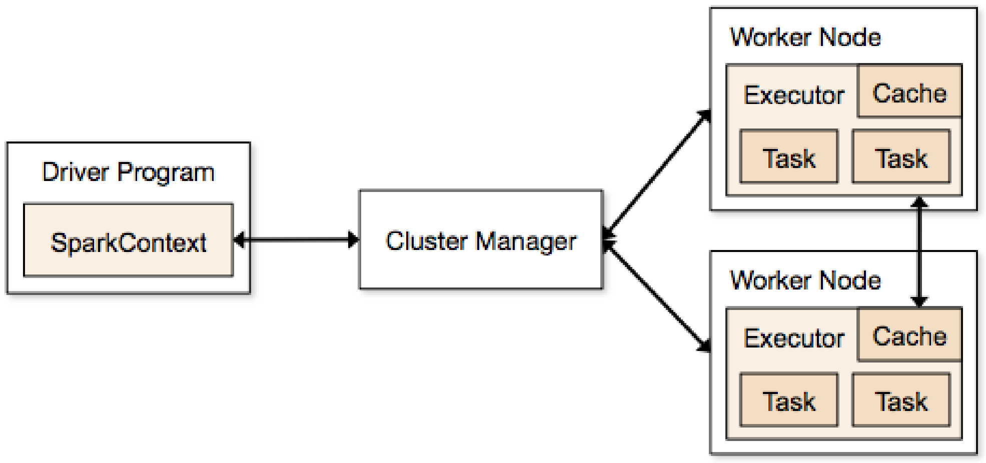


Figure 3.8: Components of a Spark cluster. The driver program contains the coordinator, *SparkContext*, responsible of making the connection between the program and the cluster manager (YARN, standalone, etc.). The worker nodes are made of an executor that runs the tasks (Image from [40]).

Chapter 4

Review of tools

Chapter 3 introduced seven platforms and tools from the Hadoop ecosystem. They were briefly discussed and an overview was given. In this section we will go back to these tools and make a study that focus on the advantages and disadvantages of them and evaluate or "grade" them according to various parameters.

4.1 Parameters

The parameters chosen are not based on any standard or any particular model. They were chosen based on the characteristics of the tools introduced before as well on the needs of Granditude. Due to the nature of our platforms and to the fact that most of them are mining/analysis tools or query frameworks, we separated the parameters in three different categories: parameters for query tools (Drill, Hive and Spark's Spark SQL), parameters for the mining/analysis tools (Flink, Mahout, Pig and Spark) and for both of them (Solr is in this category). The next list introduce the parameters.

- Parameters for query tools
  - *Ad-hoc* queries
  - Query language
  - Columnar storage support
- Parameters for mining/analysis tools

  - Batch and streaming jobs
  - Interactive and batch mode

- Parameters for both query and mining/analysis tools
  - Supports local mode
  - Extensibility
  - Data sources
  - Flexibility
  - In what programming language is it written?
  - Result format
  - Ease of use and learning

4.1.1 Parameters for query tools

Ad hoc queries

An ad hoc query is a query written for a specific purpose. This type of query is the most common one, usually written to get data needed on the moment. Of the three query tools we are studying (Drill [1], Hive [41] and Spark SQL [38, p. 181]), all of them support ad hoc queries. Moreover, Spark Streaming also support ad hoc queries on the streams. However, some of the tools have a level of flexibility at the time of executing these queries since some of them do not require the schema beforehand for some cases. In general these tools employ "schema on read" which means that the schema of the data is discovered while it is being readed from the source. In the case of Drill and Spark, both of them can infer the schema of a JSON dataset, thus we could execute queries on those datasets without manually describing how the data is. Hive does the same with Avro datasets because the schema is embedded in the data.

Query language

In this parameter we will give an overview of the query language used by Drill, Hive and Spark SQL. We will introduce the data types of the language, some of the built-in functions, operators and state whether the language follows a standard revision of SQL (and to what extent) or if it is an SQL-like interface.

Drill Apache Drill query language is based on the 2003 revision of SQL, providing a syntax that complies with ANSI SQL standard. It introduces five data types and six types of operators and five categories of built-in functions. The next list introduces the five data types with some examples.

Character types

- `CHAR`
- `VARCHAR`

Date and time types

- `DATE`
- `INTERVAL`
- `TIME`
- `TIMESTAMP`

Numeric types

- `BIGINT`
- `DECIMAL`
- `DOUBLE`
- `FLOAT`
- `INT`
- `SMALLINT`
- `REAL`

Complex types

- Map: set of key/value pairs.
- Array: List of values.

Additional to these data types, it also offers a Boolean type. On the operators side, Drill introduces comparison, logical, math, pattern matching, string and subquery operators. Some of theses are: `<, IS NULL` (comparison), `AND, NOT` (logical), `LIKE, SIMILAR` (pattern matching) and `EXISTS, IN` (subquery). Drill includes many built-in SQL functions that are categorized into: aggregate, aggregate statistics, convert, nested data and scalar functions. Below are some examples.

Aggregate functions

- `avg(expression)`: Returns the average.
- `max(expression)`: Returns the maximum value.

Aggregate statistics functions

- `stddev(expression)`: Returns the standard deviation.
- `variance(expression)`: Returns the variance.

Convert functions

- `CONVERT_FROM(string bytea, src_encoding name)`: Encode data.
- `CONVERT_TO(string text, dest_encoding name)`: Decode data.

Nested data functions

- `FLATTEN(array)(string bytea, src_encoding name)`: Flatten the elements in an array into individual records.

Scalar functions

- `length(string)(string bytea, src_encoding name)`: Returns the length of a string.
- `to_char(int, text)`: Data type formatting from int to text.

Hive Apache Hive's query language, HiveQL is an SQL dialect, similar to MySQL, and unlike Drill SQL, it does not follow any particular revision of ANSI SQL [42]. In the upcoming paragraph we will introduce some of the characteristics of HiveQL starting with a list of the supported data types.

Date and time types

- `DATE`
- `TIMESTAMP`

Misc types

- `BOOLEAN`
- `BINARY`

Numeric types

- `INT`
- `TINYINT`
- `SMALLINT`
- `BIGINT`
- `FLOAT`
- `DECIMAL`
- `DOUBLE`

String types

- `STRING`
- `VARCHAR`
- `CHAR`

Complex types

- Arrays: `ARRAY<data_type>`
- Maps: `MAP<primitive_type, data_type>`
- Structs: `STRUCT<col_name : data_type, ...>`
- Union: `UNIONTYPE<data_type, data_type, ...>`

Hive operators are divided in four categories: arithmetic, logical, relational and operators over complex types. Some of these are: `AND` or `&&,` `NOT` or `!` (logical operators), `=` `,` `!=,` `<,` `>` (relational operators), and `A[n]` (operator for complex type) for returning the nth element on an array.

Hive's list of built-in functions includes functions for different uses such as aggregate functions, mathematical functions, collection functions like `size(Array<T>)`, type conversion functions, date functions, conditional functions, string functions and table-generating functions that transforms an input row into multiple output rows.

Spark SQL Spark SQL's query language supports a subset of the third revision of SQL, SQL-92 [43]. Besides its own SQL language, due to its compatibility with Hive, Spark SQL supports HiveQL queries within a Spark program, using most of the features discussed for the platform.

In terms of operators and built-in functions, most of the functionality available for RDDs, also an implementation for Spark SQL and its DataFrames.

The following list presents Spark's SQL supported data types.
Numeric types

- `ByteType`
- `IntegerType`
- `ShortType`
- `LongType`
- `DecimaType`
- `DoubleType`
- `FloatType`

String type

- `StringType`

Binary type

- `BinaryType`

Boolean type

- `BooleanType`

Datetime types

- `DateType`
- `TimestampType`

Complex types

- `ArrayType(elementType, containsNull)`: Sequence of elements of `elementType`. `containsNull` indicates whether elements of the array can be `null`.
- `MapType(keyType, valueType, valueContainsNull)`
- `StructType(fields)`, where `fields` is a sequence of `StructField(name, dataType, nullable)`.

Columnar storage support

A columnar database is a DBMS that stores the data in columns instead of rows of data, resulting in several advantages such as better performance when applying aggregating functions. Of our three query tools, all of them supports columnar storage. On Hive and Spark SQL this is done with the support of Apache Parquet, a columnar storage format made for the Hadoop ecosystem.

4.1.2 Parameters for mining/analyzing tools

Batch and streaming jobs

In addition to the typical batch jobs performed by most tools, we are looking for those which also support data analysis of real-time streams of data. Of the tools we are studying for mining and analyzing data (Flink, Pig and Spark), Flink and Spark support streaming jobs. Flink streaming API is called Flink Streaming and it is an extension of the core Flink API and Spark streaming platform is Spark Streaming which is also an extension of the core Spark API. By being an extension of their respective core APIs, both platforms are able to execute the same high-level functions of their batch counterpart; in the case of Spark, GraphX and MLlib can be used on data streams.

Interactive and batch mode

For our next parameter, we are interested in knowing which one of the platforms supports interactive execution mode as well as batch execution mode. Interactive mode provides a way of performing quick jobs as well as testing the environment and debugging without the need of having to write a complete script while in batch mode, the jobs are submitted usually through a script. Of our three analysing tools, all of them supports batch mode, while Pig and Spark also provides support for interactive mode. Pig interactive mode allows the execution of Pig commands and Pig Latin statements and Spark interactive mode supports interactive analysis in either Python or Scala.

4.1.3 Parameters for both query and mining/analyzing tools

Supports local mode

Being able to run the platform in local or non-distributed mode is useful for testing the environment and the jobs to perform without having to set up a cluster.

Starting with the query tools, Hive supports Hadoop's local mode setup where it can be run in a single node and Drill also has local mode called *embedded mode.*

The mining and analysis tools are also able to run in non-distributed mode. Flink's default configuration make it able to run in single node setup out-of-the-box. When Spark is running in local mode, it assigns a worker to each CPU core, making more pseudo-distributed. Pig supports two execution mode, local mode and mapreduce. When running on local mode, it uses a single machine and the local file system while in mapreduce mode a Hadoop cluster, even if it is a single node cluster, is needed.

As for the remaining platforms, Solr and Mahout, Solr supports local mode and Mahout can be used on any Java program on a single machine.

Extensibility

In this category we are looking for those platforms that support custom user-defined functions (UDF) written in a language other than the ones used for the applications. From our list of tools, five of them, Drill, Hive, Pig, Solr and Spark, support custom user-defined functions. Drill provides a Java API that contains two interfaces to write simple functions that operates on a single row and produces another single row, and aggregate functions that operates in multiple input rows and produces a single one. Hive also provides a Java API and Pig also provides a Java, Python, Ruby, Groovy and JavaScript API as well as a repository for API called *Piggy Bank*. Lastly, the query component of Spark, Spark SQL is compatible with Hive UDFs.

Data sources

In this parameter we list all the possible external sources from where each tool can acquire data.

- Flink: HDFS, HBase, Amazon S3 and the local file system.
  - Flink Streaming: Kafka, RabbitMQ, Flume, Twitter Streaming API, file system sources and socket stream.
- Drill: HDFS, HBase, Amazon S3, Hive tables, local file system and NoSQL databases such as MongoDB and MapR-DB.
- Spark: HDFS, HBase, Cassandra, Amazon S3, local file system, OpenStack Swift
  - Spark Streaming: Kafka, Flume, HDHS, Amazon S3, Kinesis, Twitter Streaming API, socket connections, ZeroMQ and MQTT.
  - Spark SQL: Hive tables.
- Hive: HDFS, HBase, local file system and Amazon S3.
- Pig: HDFS, local file system and Amazon S3.

Flexibility

How flexible the platform is in terms of the type of the data supported, for example, structured data, JSON, XML among others as well as the programming languages supported by the tool to write applications.

Because all of our tools are compatible with the Hadoop ecosystem, they mostly support Hadoop's input formats which are text files and binary files based on the interface `InputFormat`. The formats of the content of the file, does not matter and it can be in JSON, XML, unstructured, etc. The programmer is the responsible of how to parse and work with the content of the file. However, some tools include built in functions and deserializers to parse them. Flink includes a built-in parser for CSV and JSON files and a `FieldParser<T>` class for parsing a sequence of bytes into a type T, where is can be int, float, string among others. In the case of Spark, Spark SQL is able to read Parquet files as well as JSON files and parse them automatically. Pig also includes a built-in JSON parser. Moving on to the query tools, Hive include several built-in DeSer (deserializer/serializer) for the following formats: ORC, RegEx, Thrift, Parquet and CSV. Drill, on the other hand supports: CSV, TSV, PSV, Parquet and JSON.

Another requirement for this parameter are the programming languages out-of-the-box supported to write applications on the mining tools.

- Flink: Java and Scala.
- Pig: Pig Latin
- Spark: Java, Python and Scala.

Result format

How does the platform return the results? How is the output? We will look at the ways in which the platform returns the data and if it can be modified.

- Flink:
  - Formats: Avro, binary, CSV and plain text.

  - How to change it: Modified by changing the output format when calling the `write` function or any of the functions `writeAs....` Example ... `output.write(wordsOutput, CsvOutputFormat())` or `output.writeAsCsv(...)`

- Hive:
  - Formats: Plain text, Avro, Parquet, HBase, HiveBinary, ORC and RCFile.
  - To modify it add the following command in the query, `OUTPUTFORMAT output_format_classname`

- Spark:
  - Formats: Plain text, JSON, Hadoop's SequenceFile, Parquet and as an object file using serialization.
  - How to change it: To modify it, use any of `saveAs...` functions on the RDD object. Example, `rddObject.saveAsTextFile(...)` or `rddObject.asJSON()`.

- Drill:
  - Formats: The output format of a table can be in: CSV, Parquet or JSON.
  - How to change it: To modify the output format of a table, change the store.format option like this example: `ALTER SESSION SET ‘store.format‘=’json’`.

- Pig:
  - Formats: Plain text and JSON.
  - How to change it: Use the following format to store a table in JSON format `STORE first_table`.

- Solr:
  - Formats: The result of the search can be in CSV, JSON, PHP, XM, XSLTL. It also provides support of making the output ready for evaluation for PHP, Python and Ruby
  - How to change it:To change it, simply write the desired format in the `wt` parameter on the search query.

Programming language(s) is which it is written	
Tool	Programming language(s)
Hive	Java
Drill	Java
Flink	Java and Scala
Spark	Java, Scala and Python
Pig	Java
Mahout	Java
Solr	Java

Table 4.1: Programming language(s) is which it is written

In what programming language is it written?

The idea behind knowing the programming language in which each tool is written is to have a more in-depth idea of how it works, the advantages and disadvantages and since they are open source, it allows us to modify the code if it is needed to suit our needs. Table 4.1 presents the programming languages in which each platform is written.

Ease of use and learning

In this category, we will be discussing the documentation (website, books, guides, tutorials, presentations) available for each tool for learning how to use it, an overview about the installation process and about the community.

Flink The documentation on Flink's website includes a programming guide for both Java and Scala, guides for its streaming component and Spargel API as well as instructions about how to install it on several OS. Besides this, finding external resources was such as books was not possible. For deploying on a single machine, Flink comes already ready to be executed out of the box, requiring only a version of Java higher than 1.6.x and the environmental variable `JAVA_HOME` pointing to the correct location. To install it in a cluster, the package needs to be unpacked on the same path on all the machines, then the user needs to chose one of the machines as a master node and there, add the IP or hostnames of the worker nodes and to install Flink on YARN only a command line needs to be executed. About the community, there is a mailing list for news and for user support among others, an IRC channel.

Lastly, the Drill team have given several talks introducing Flink at several locations such as Palo Alto, CA and Stockholm.

Drill Drill's website contains a number of tools to facilitate the learning process of an user. It has several videos that gives an overview of the system, how to use it with Hadoop, how to configure it and more that complements its wiki guide, which include topics such as how to install it, how to connect it to data sources and an extensive SQL reference guide that introduces the data types, operators, functions, commands and the reserved keywords of the platform. To run it on a single machine, a single command needs to be executed from installation directory.F or a clustered environment ZooKeeper is a requirement and the process involves creating a unique cluster ID and assign it and all the nodes followed by proving ZooKeeper the host names and port numbers. On the community page they have listed all the events and meetup and workshops and conferences, their mailing lists and a link to their Slideshare which has several presentations. In addition to this, they do a Google Hangout every month.

Spark Besides its website, documentation about Spark can be found in different sources and formats. At the time of writing, there are two books available about learning Spark, one of them is considered a guide for the platform while the other is for doing advanced and more complex analytics on it. In addition to this, there is an upcoming course in the massive open online course (MOOC) platform, edX about Big Data using Spark. Installing Spark is a straightforward process, with JDK 6 as the only prerequisite. Upon downloading and unpacking the package the user just need to run one of two commands to start it (this apply for standalone and cluster mode). One of the command it so start the cluster manually and the other to start it automatically. Regarding the community and their activities, they do monthly meetups in several locations, there are mailing lists and a series of yearly summits that have been celebrated since 2013. The content of the summits include development of the tool and training sections.

Hive and Pig These two platforms have been part of the Hadoop ecosystem since its inception. Because of to their maturity and longevity in the market, the amount of information about them is abundant and accessible. Supplemental to their respective websites, which include guides about their languages, installation, wikis and resources for contributors, there are several books about them related to how they work, and how to build applications. In addition to this, there are private sources and training sessions. The installation process for both tools is fairly similar, it involves setting an environmental variable to the path where it was installed, add said variable to the PATH; to run them on Hadoop, Hadoop needs to be on the PATH, otherwise export an environmental variable. In the case of Pig, the variable is `HADOOPDIR` and it should point to Hadoop's hadoop-site.xml file and for Hive, `HADOOP_HOME` to Hadoop's installation directory. In terms of the community and workshops, both tools are present during the Hadoop Summit where sections are given, besides this there are several Meetup groups.

Solr Over five books, presentations, videos, guides, tutorials, reference guides and Meetup groups are some of the tools available for learning Solr. The installation process varies depending on what the user want to do; for a development environment just run the command `bin/solr start` on the installation directory to start the Solr server. However, for a production environment the user should run an installation script, and tune several settings such as ZooKeeper for running Solr in distributed mode. The only requirement is JRE 1.7 or higher.

Mahout For each one of the algorithms available in Mahout, the official website offers a brief introduction to the algorithm (that in some cases includes math definitions), followed by an example. Outside of the website, there are books, tutorials and presentations that are mostly about learning Machine Learning using Mahout. Respecting the community, through Meetups they offer panels giving insights about how to build applications using Mahout. Regarding the installation process, since Mahout is a library, to use it in Hadoop or any Java environment like Eclipse, the user needs to download the source code and compile it using Maven.

4.2 Summary

Chapter 3 introduced seven tools that are the main topic of this work. There a brief overview about them was given, followed by listing their main features and exposing their programming model and internal functionality. This description of the tools was continued in this chapter where they were further studied and analyzed around a number of parameters. In this section we will continue and finish the work we started on chapter 3 by summarizing and discussing the findings regarding the following criteria: usage, use cases and data environment.

4.2.1 Usage

Under this criteria, usage, is it of our interest to find out the kind of user that these tools are aimed or suitable for. In the same way these platforms can be used for several purposes (as we will see in the next section), the form they are used and the results produced by them might be of interest to one or more groups of users. Here we will classify the platforms under one of three different possible users.

- Data Scientist
- Database User
- Business Analyst

The data scientist, a person that we will define as someone who analyze, explore and models data with the goal of reaching certain conclusions about it. The second user is a database administrator or database user, a person who is familiar with how a database (in particular a relational DBMS) works and has a certain knowledge about the internal structure of a database, their query language and a general idea of how it works as a whole. The last user is a business analyst, a person who is mostly interested in using the platforms to acquire insight about a certain business. Regarding the last user, a business analyst, we should mention that the interaction between them and these platforms is usually done through a third-party program (a business intelligence platform), meaning that the user will not work directly on it; depending on the kind of analysis, the business analyst should have a certain knowledge about the platform he is interacting with. This knowledge may include, understanding of the programming language(s) or an overview of how it works, otherwise if the person do not have the required background, it might be difficult to use them.

Drill

Of the three possible roles described for this criteria, someone with a database background seems to be the most appropriated person for using Drill. Its support of an ANSI SQL query language, makes it almost possible for someone who has worked previously with databases, to use this platform without the need of learning another type of SQL. Drill also features a connection using a ODBC or JDBC plugin, allowing BI platforms such as QlikView to perform queries on Drill, thus making Drill an attractive platform also for business analysts.

Flink

Flink is an engine for data processing, therefore it fits in our definition of what should be a tool aimed to a data scientist. It does not support a query engine nor ODBC/JDBC connectivity, so based on our definitions it does not fit the role of the other two users. However, on its source code we found code related to JDBC but further documentation could not be found.

Hive

Based on our users, Hive fits the role of a platform of a database user. It include an ODBC and JDBC plugin for connections between the platform and external BI platforms, hence it fits our definition of a tool that could be used by a business analyst.

Mahout

From our list of possible users, Mahout's machine library fits the role of a data scientist. Being a library means that Mahout can be used in other platforms or Java projects, so it uses could be extended to other users not described here.

Pig

Pig share the same characteristics as Flink. It does not include an query engine nor supports connectivity to any external BI tool, on the contrary, it connects to other tools through a JDBC connection. This makes Pig a tool for a data scientist, by our definitions.

Solr

Solr, a search platform , is the tool from our study that does not suit any of the roles mentioned here as good as the other tools. Being an information retrieval platform, it goes more with the line of work of a software architect or system administrator [44, p.17 18].

Spark

Spark could fit the role of all the users defined at the beginning of this section. It is a data processing engine like Flink, fitting the data scientist role. Moreover, its SQL API, Spark SQL might be of interest to a database user and so it JDBC and ODBC plugin to a business analyst.

4.2.2 Use cases

For the next criteria, use cases, we came up with a list of five possible use cases, based on the study of the tools done in chapter 3 and at the beginning of this chapter. In this section we will classify our seven platforms in one or several of the uses cases according on whether they could be used for those purposes or not. The following list shows the use cases.

- Ad-hoc queries
- ETL data pipelines
- Machine learning algorithms
- Batch/streaming oriented
- Provition of fast access to data

Drill

The purpose of Drill inside the Hadoop ecosystem is to serve as a query layer for performing low-latency, schema-less and *in-situ* ad-hoc queries employing full table scans from a number of data sources.

Flink

Flink, an alternative to Hadoop's MapReduce, serve as a data processing engine. Besides being used for traditional batch processing, it is also capable of performing operations over streams of data using an extension of its API, Flink Streaming. Thus, out of our five different use cases, we classify Flink as a platform that is both batch and streaming oriented.

Hive

Hive is suited for interactive data browsing using ad-hoc queries and batch processing on relatively static or immutable data. In section 4.1.1 we talked about ad-hoc queries using Hive and its ability with some data formats without needing the schema. However, the ad-hoc queries in Hive comes with disadvantages and advantages. On one side, Hive is not a full database, thus it has some limitations such as the inability of performing online transactions or the support of record-level updates [23, p. 2]. On the other side, an advantage of Hive's queries is the fact that they are transformed to map/reduce jobs, supporting batch processing via map/reduce using a much simpler way through HiveQL. However, this is also disadvantage because even the simplest job will cause unnecessary overhead to the system and high latency.

Mahout

Mahout is a platform for data analysis that provides the support needed for solving machine learning problems in a scalable way. It mainly covers the use cases of problems involving collaborative filtering, clustering, classification and dimensionality reduction.

Pig

Pig is used for two main purposes: ETL data pipelines and data analysis, with ETL being the principal use case [30, p. 7]. In an ETL Pig program, the first step is loading the data from the HDFS or any other of Pig's data sources, followed by operating on said data and then load it back to a destination. The second use case to discuss is Pig's ability to perform data analysis. Pig has several advantages over a regular MapReduce job, it provides most the operations while being simpler than and shorter than an equivalent

MapReduce job. On the other hand unlike Flink and Spark, Pig is batch oriented.

Solr

Solr, an enterprise search server does not suit any of the roles mentioned here as good as the other tools. It is not directly linked with data analysis, instead its role is more on the lines of information retrieval and searching so it covers our need of having a tool for fast access to specific data. In some sense it could be categorized as a query tool, but unlike the Drill and Hive which focuses on queries for structured and well-organized data, Solr supports NoSQL queries for data of unstructured nature (usually text).

Spark

Spark serves as a general-purpose framework for data analysis. It is made of five different components that covers a range of use cases. The first of these components is Spark Core, the main foundation of Spark, responsible of providing the APIs and functionalities for the platform. Spark SQL is Spark's API for working with structure data which also provides ad-hoc queries like Flink and Hive. Additional to its capacity of performing batch jobs, Spark also covers the necessity of streaming data analysis through Spark Streaming. Spark also covers machine learning tasks through its library MLlib that include algorithms for tasks such as classification, regression and clustering. The last component is GraphX, a library somehow similar to Flink's Spargel for manipulating graphs. In some way, Spark covers most of the use cases the other tools also cover, with the exception of searching, resulting in the advantage of using just one tool instead of relying on many of them, however this does not means that it does a better work in every category. For example, Mahout includes some algorithms not covered by MLlib and Spark SQL fits more the role of a query engine, like Flink instead of a warehouse like Hive.

4.2.3 Data environment

For the last criteria, data environment, our goal is to find the data environment in which our platforms work. In section 4.1.3 we introduced different formats files that are used and/or produced by each one of the platforms.

In this section we will use those formats and look at them as a whole set to classify the data environment of the tool in one or more of the following categories:

- Structured: Data based on schemas.
- Semi-structured: Data with internal structures like XML or JSON.
- Unstructured: Video, images, documents.
- Data lake: Mix of all types of data.

Before we start, we wish to note three things. First, in this section we are assuming we are using these platforms with HDFS as the data storage layer. We also wish to remind that Hadoop's HDFS stores anything that can be transformed into binary, however, for some file formats, in particular for text files, it offers built-in functions to read them as they are. For other type of files like images and videos, a custom `SequenceFile` or `InputFormat`, `OutputFormat` and `RecordReader` needs to be written, meaning that in theory mostly every file can be read by HDFS and that makes it a data lake with data that can be used by all of our platforms. The second point is about plain text files, which for the purpose of this work we defined as a text file whose data is not structured and the content can be anything, for example a blog, a note or a dump file; all the platforms discussed here support plain text files, thus we will not mention this fact on the discussion. The third point we wish to state is that the following discussion might get repetitive as a result that the tools share many similarities in terms of the data formats it supports.

Drill

Drill's main support of files lies in the category of semi-structured data. Of the four type of data supported mentioned before in section 4.1.3, four of them are categorized semi-structured by our definition: CSV, TSV (tab-separated), PSV (pipe-separated) and JSON. The remaining one, Parquet is a structured file format that incorporate the schema as part of the file. Besides this, Drill also works on the tables it creates, which is another example of a structure environment; analogous to this, it is able to use Hive tables (structure data) as an external data source.

Hive

Out-of-the-box Hive supports distinct file formats using its SerDe (serializer/deserializer) interface for IO. Formats like Parquet, Thrift and ORC incorporate the schema of the data defined in the file. This built-in schema satisfy our definition of a structured file format. Hive include one more SerDe for RegEx, a data format that we considered to be semi-structured due to the fact that it internal structure defines the object, in this case the regular expression string.

Flink

During Flink's examination, we found out it includes a built-in parser for CSV and JSON files and a class `FieldParser<T>` whose function is parsing a field from a sequence of bytes where `T` is the type to be parsed. Given these parsers and the class, we concluded that Flink's data environment operates on semi-structured data.

Mahout

Mahout does not apply in this criteria. Its supported files are the one supported by the environment where it is used.

Pig

Besides being able of working with files stored in HDFS and plain text files, Pig's additional support of serializer/deserializers and parsers just extend to the semi-structured file format, JSON.

Solr

The typical use of Solr is to index structured data such as databases tables or semi-structure data like XML or CSV files. However, it can also be used for unstructured and binary data like images and videos [32, p. 100].

Spark

Spark data environment is based mostly on structured and semi-structured data. Under structured, it support Parquet files and Hive tables. For semi-structured data, Spark supports JSON format.

Chapter 5

Web log analysis

In this chapter we introduce the practical part of this work. After working with several platforms in the previous section, we wanted to present a real application to show some of the features of one platform and see how it perform and how it is to work with it in practice.

The use case was an analysis of a web log and the analysis was done using Spark. The reason behind choosing Spark as the platform to conduct the analysis is based on the requisites established by the client. For example, one of the requirement the client mentioned was the use of Python or R as the main programming language for the task and as it was mentioned in section 4.1.3, Spark supports Python. A second requirement was running machine learning jobs over the dataset, which was done using MLlib.

In section 5.1 we will introduce the problem and our questions, followed by the setup used in 5.2, the analysis in 5.3 and a discussion of the results in 6.1.

5.1 Problem

The dataset given to us by a client of Granditude is a web log from one of Sweden's most popular newspaper. The web log contains all the traffic of March 15, 2015 from 8:00 to 9:00 Swedish time (CET). The task given by the client was to perform exploratory data analysis on the web log using Spark to showcase its capabilities and components, in particular Spark SQL and MLlib. From the original problem, we narrowed it down to focus on certain properties of the web log. These are: the location of the visitors in term of country and city, the frequency of visits, the duration of visits and the sections of the newspaper visited.

Based on those properties, we are looking to solve questions such as:

- Which are the top three countries that visits the website?
- How many unique combinations of country/city exists in the weblog?
- What are the peak times during this hour?
- Which are the most frequently visited sections by country?
- What is the average time per section?

5.2 Setup

The analysis was done in Spark 1.3.0, deployed on standalone mode using two workers of 3g of memory running on Ubuntu 14.04. The Python component, PySpark was used on batch mode. Hive 0.14.0 was also used, running on top of Hadoop 2.6.0.

5.3 Analysis of the data

In this section we will present our analysis of the data. Each question or studied case will be shown as a subsection with a description of the problem, an overview of how it was done and the result obtained.

Processing of the data

The original data was delivered in a single CSV file. The file, was imported into Hive and from here, a new table was created by converting the data to Parquet format using GZIP compression encoding. With the data divided into several Parquet files, it was imported into Spark to a DataFrame object.

Overview of data

The dataset is made of 1.7 million rows and 554 columns. The columns consists of information such as the browser used by the visitor, its language, country, visited page and more. Because Parquet is a self-describing data format, the schema of the dataset does not need to be specified.

Location of visitors

This part of the analysis is related to the geographical location of the visitors of the website. During the hour of traffic, the website received visits from 156 different countries. These visits could be anything from real users to bots or web crawlers; the data did not specified this information. Out of the top ten countries from where most of the traffic came, eight of them were European countries, the remaining two are Thailand and United States; the top three countries are Sweden, Finland and Norway with 381,574, 4,431 and 4,218 visitors respectively.

Besides the countries, we also looked for the cities of the visitors and found a total of 4897 unique combinations of countries/cities. Of these unique combinations, the top five from where most of the visitors of the website came, are the following five Swedish cities: Stockholm, Gothenburg, Varberg, Lidingö and Malmö.

To perform this part of the analysis, the dataset was filtered by those records where a variable named *hourly_visitor* was 1. This way, we only took those records where the user was visiting the website for the first time in that hour.

Most frequent section by country

As a follow up topic related to the countries, we found out which section of the newspaper (news, sports, weather among others) is the most frequent by visitors of a certain country. Of all the sections, the one about news and sports were the most visited in almost all the countries, covering 93.1%. The remaining sections are related to cars, leadership texts, quizzes, television, entertainment and viral content from the Internet.

Visitors data

Visitors data	
Parameter	Frequency
Hourly visitors	403,248
Daily visitors	289,101
Weekly visitors	289,089
Monthly visitors	35,985
Quarterly visitors	29,018
Yearly visitors	29,018
Users visiting the website for first time	81,085

Table 5.1: Visitors data

The weblog contains several columns about the visits of an IPs to the website. Some of the columns inform if it is the first time the website receives a visit from an unique IP during a period of time, another column specify if it is a new visit and the visit number from that IP. The following list explains this with more detail and table 5.1 summarize the results of the findings.

- *hourly_visitor, daily_visitor, weekly_visitor, monthly_visitor, quarterly_visitor, yearly_visitor*: The value of this column is 1 if it the first visit of a certain IP during the current hour/day/week/month/quarter/year.
- *new_visit*: The value of this column is 1 if it a new visit; this is the event of visiting the website. For example, supposed a person access a website and reads two articles, then out of the three different records in the web log (one per page viewed), one of them will have *new_visit* set to 1 and the others two, set to 0. If the same user visits the page during another section, then *new_visit* will be 1.
- *visit_num*: This column displays the number of visits registered by an IP. It is not known exact moment when this counter started.

Using this information we looked for whether the new visitors (*new_visit* = 1) goes to the homepage of the newspaper or to an specific article. We found out that out of the 81,085 new visitors, 39,266 went directly to read an article and 37,344 went to the newspaper's homepage. The rest of the sections visited are mostly distributed between video, audio content and not found pages.

The countries of the new visitors was also explored. We identified that most of these come from Sweden with 73,525 (90.68%) new visitors, followed by Finland with 1,542 (1.90%) and Norway with 1,181 (1.46%). By using these results and findings explained in the last paragraph we discovered that from new visits from Sweden, 35,403 (48.15%) accessed an article directly while 33,349 (45.36%) went to the homepage. Regarding Finland, 755 (48.97%) of the new visitors went to an article and 715 (46.37%) to the homepage and respecting Norway, 615 (52.07%) accessed an article and 523 (44.28%), the homepage.

Times of visit and duration of visits

This section of the analysis targets the time of visits to the website and the duration of the visits. The first inquiry we wanted to investigate in this section was the peak period of time that received the most visitors. We found out that from the start of the hour up to the minute 54, the average number of visitors per minute was 27274.40. However, during the last minutes of the hour (from minute 55 to 59), the average is 37780.20. Figure 5.1 shows this. From here, we continued working in this and discovered that an article about a Swedish football player, was most visited during that period with a peak of 5977 visits during minute 56.

The second issue studied in this section was the average time spent per section of the newspaper. Because the dataset just contains the hit time of the link, we needed at least two different viewed pages, this way we were able to subtract the hit time of the second page from the hit time of the first viewed page to obtain the time spent in that first article. This means that if the user just visited one page, we were unable to obtain the time spent in that page or if it visits more than one page, we were not able to obtain the time spent in the last visited page. Another detail about this is that we did not differed whether the user is visiting the front page of a section or reading an article belonging to that section. For example, if user A spent 30 seconds in the sports front page and user B spent 45 seconds reading an article belonging to the sport section, the average time spent in the sport section would be 37.50 seconds. Table 5.2 shows some of the sections and the average duration of visits.

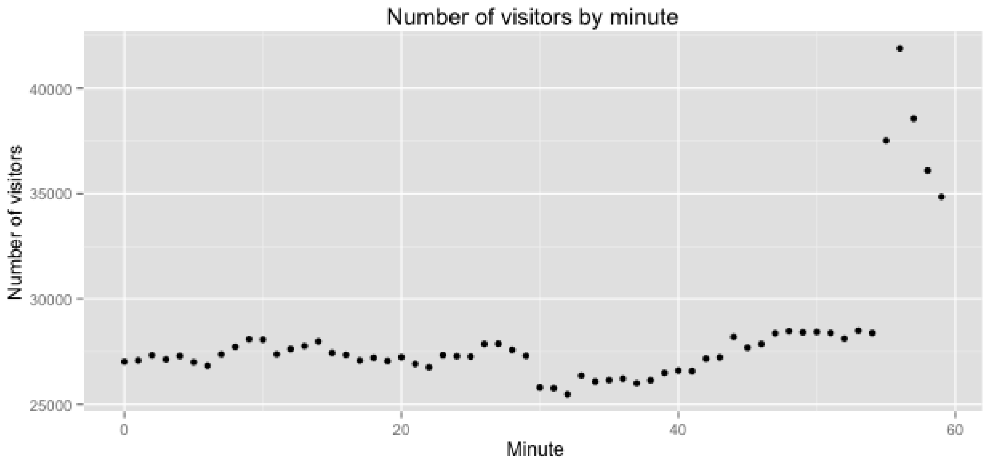


Figure 5.1: Visitors by the minute. From 8:00 to 8:54 the average number of visitors per minute was 27240.40. From 8:55 to 8:59, the average increased to 37780.20.

This test was repeated a second time and unlike the previous one, the results were filtered to use those rows where an article is accessed and ignore the rows in which a front page of a section is accessed. Table 5.3 shows the average time spent in the same sections as table 5.2 with the exception of the website's homepage and the TV/videos section because they are not categorized as 'article'.

Classification using Machine Learning

During the last part of the analysis we introduced Spark's machine learning library, MLlib, to investigate if it is possible to predict the section of the newspaper read by a visitor from Sweden based on whether it is a new visit, the city of the user, the time spent in the page and whether the user is accessing the site through the full site or mobile site. Two models were built using two different multiclass classification algorithm, Naive Bayes and decision tree. For both models we split the dataset into a training set made of 70% of the data and a test set made of the remaining 30%. To reduce the amount distinct features, instead of using all the Swedish cities from the dataset, we chose the top 30 with the most visitors.

Average time spent in section	
Section	Average time in seconds
Health section	100.42
TV/videos section	86.31
Culture section	79.79
News section	69.59
Website's homepage	68.85
eSports (competitive videogames) section	65.37
Sports section	61.61
Entertainment section	56.80
Weather section	53.32
Latest news	51.59

Table 5.2: Average time spent in section

Average time spent in section (article)	
Section	Average time in seconds
Health section	222.61
Culture section	108.17
News section	114.67
eSports (competitive videogames) section	139.83
Sports section	109.48
Entertainment section	95.08
Weather section	42.79
Latest news	78.77

Table 5.3: Average time spent in section (article)

A naive Bayes is a probabilistic classifier that assign a score to each vector based on how well it belongs to each of the classes based on a linear function of the features [38, p. 227]. In Spark, the naive Bayes classifier takes as a parameter a smoothing constant, which we left it with the default value of 1.0. Three different runs were performed, obtaining the following accuracies when tested against the test set: 59.98%, 60.05% and 59.89%, an overall of 59.97%.

The second model used, a decision tree, is a classification model represented by a tree where at each node, a binary decision is made based on the data and each leaf represents a class. Our decision tree used Gini impurity as the impurity measure, had a max depth parameter set to 5 and max bins set to 32. Like the naive Bayes model, three runs were performed, achieving the following accuracies when tested against the test set: 40.02%, 39.95% and 40.11%, an overall of 40.03%.

Chapter 6

Conclusion

In this final chapter, we will discuss the results about the web log analysis that were presented in the chapter 5. The discussion will be followed by the limitations and issues encountered during the project. At the end of the chapter, we will state and give our input on what could be done in a future project.

6.1 Analysis discussion

In this section, we will discuss the results obtained in the analysis followed by our opinion or interpretation of them.

In the first part of the analysis we worked with the geographic location of the visitors of the website. We discovered that out of the top 10 countries, 8 of them belonged to Europe, with Sweden, Finland and Norway being the first three. This result was the expected one considering that the website is about Swedish news, so Nordic countries were expected. Regarding the top towns, the results were satisfactory. Prior to the analysis we were expecting to see on the top five, some of the Swedish biggest cities such as Stockholm, Gotherburg and Malmö. In the next section of the study, we kept working on the data related to the countries. On that instance we introduced the sections of the newspaper to the analysis and we looked for the most frequent section of the newspaper by country. The results were very similar across every country, with the news and sports section being the most frequent with over 90%.

The next topic of the analysis focused on the visitors data. Our intent with this was to have an understanding about how frequent an user visits the website and what sections are visited by those users who are accessing the website for their first time. From table 5.1 we can see that during that hour, the website received over 400,000 unique visitors. Of those 400,000, around 100,000 already visited the website during that day. We were expecting this number to be large as a result of the timeframe of the analysis (Monday, 8:00 to 9:00). At those times people are usually commuting to work, having breakfast or starting the day, activities where is frequent to read the newspaper. About the daily and weekly visitors counter, we were expecting this number to be the same for the reason that Monday is the first day of the week, thus is a person is visiting the website for the first time during that day, it implies that it is also visiting the website for the first time that week. Yet, the counters are very similar, showing a difference of 12, thus our assumption was partly correct. Lastly, the yearly and quarterly visitors counter shows the same number because March belongs to the first quarter of the year.

As a continuation of the visitors data analysis, we discovered that from all the users visiting that website for the first time, around half of them go to the homepage of the website and the other half go to an article. We went further into this and found that the top three countries with the greater number of new visitors, once again, Sweden, Finland and Norway, also showed the same behavior.

During the study of the peak times, we found interesting the fact that during the last minutes of the hour, the amount of visitors that accessed the website increased by around 10,000. Upon further analysis, we were able to find the exact article that was being viewed, however no other information could be found about this. We believed one possible reason for this behavior is that said article was linked from another location such a social network.

In table 5.1 and 5.2 we showed the average time spent per section of the newspaper. The first table shows the time for both an article and the front page of a section and the second table, the time average time an user spent in articles. The results obtained indicate that the users usually spent more time in an article than in a front page of a section of the newspaper. The only section where this pattern was not observed, was the weather section. Usually, the home page of a weather section has all the information related to the weather on it, without the need of looking at an specific article, so we concluded that this result is valid.

The results gained in the machine learning part of the analysis were quite unsatisfactory with a 60% accuracy for the naive Bayes model and 40% in the decision tree. These numbers could mean one of two things. First, there might be indeed a certain pattern related to the visits in the data. Second, if we recall the analysis about the peak times and the most frequent sections, it was observed that sports and news were the most visited section, so there is a chance that our models might be biased towards these sections and as a result, the records will be classified as news or sports, achieving a high accuracy.

6.2 Future work

Regarding the first part of the project, the academic survey about the platforms, the work done on this thesis can be extended in different ways. The platforms presented were delimited to those that focus primarily in data analysis and query. However, this work could be extended with platforms that aim different purposes than those presented here, such as security for Hadoop, platforms for streams of data, data ingestion platforms, data visualization or platforms that monitors or manage the cluster. Also, the list of parameters presented in chapter 4 could also be extended to investigate other topics such as the accessibility of the platform (who can access what) and user interface (if it offers one).

The web log analysis could also be improved and extended using other of the features present in the dataset. During our investigation we only used around 50 columns out of the 554 columns that are exist in the dataset. The machine learning section could be improved by changing the parameters of the model and with further pre-procession of the data. Another part of the analysis that could be continued in a future work is the peak times section. The investigation showed that during a certain of time, the number of visitors increased and we never found an answer or a reason for this behavior. Lastly, with a bigger dataset, one that span for at least one week, we could investigate the peak times across one week, most frequent articles per day and the number of visitors per day.

6.3 Limitations

The main issue encountered during the project was regarding the setup used for the analysis. The original plan was to access the data using the HiveQL component of Spark SQL. However, the version of Spark used, was not compatible with Hive 0.14.0.

List of Figures

List of Tables

References

[1] M. Hausenblas and J. Nadeau, “Apache drill: interactive ad-hoc analysis at scale,” *Big Data*, vol. 1, no. 2, pp. 100–104, 2013.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: cluster computing with working sets,” in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[3] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, “Building a high-level dataflow system on top of map-reduce: the pig experience,” *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.

[4] V. Markl, “Breaking the chains: On declarative data analysis and data independence in the big data era,” *Proceedings of the VLDB Endowment*, vol. 7, no. 13, 2014.

[5] T. White, *Hadoop: The definitive guide.* " O’Reilly Media, Inc.", 2012.

[6] D. Borthakur, “Hdfs architecture guide,” *Hadoop Apache Project*, p. 53, 2008.

[7] Rack-aware replica placement. [Online]. Available: https://issues.apache.org/jira/browse/HADOOP-692

[8] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, “Apache hadoop yarn: Yet another resource negotiator,” in *Proceedings of the 4th annual Symposium on Cloud Computing.* ACM, 2013, p. 5.

[9] Apache Hadoop NextGen MapReduce (YARN). [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[10] Mapreduce Tutorial. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/\hadoop-mapreduce-client-core/MapReduceTutorial.html

[11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] Apache Drill. [Online]. Available: http://drill.apache.org/

[13] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[14] Apache HBase. [Online]. Available: http://hbase.apache.org/

[15] Amazon S3. [Online]. Available: http://aws.amazon.com/s3/

[16] Mongodb. [Online]. Available: http://www.mongodb.org/

[17] Apache ZooKeeper. [Online]. Available: http://zookeeper.apache.org/

[18] Architectural Overviewl. [Online]. Available: https://cwiki.apache.org/confluence/display/DRILL/Architectural+Overview

[19] Apache Flink. [Online]. Available: https://flink.apache.org/

[20] Spargel. [Online]. Available: http://ci.apache.org/projects/flink/flink-docs-master/libs/spargel_guide.html

[21] Flink's process. [Online]. Available: http://ci.apache.org/projects/flink/flink-docs-release-0.7/internal_general_arch.html

[22] Apache Hive TM. [Online]. Available: https://hive.apache.org/

[23] E. Capriolo, D. Wampler, and J. Rutherglen, *Programming hive.* " O'Reilly Media, Inc.", 2012.

[24] A. Floratou, U. F. Minhas, and F. Ozcan, "Sql-on-hadoop: Full circle back to shared-nothing database architectures," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, 2014.

[25] Hive system architecture. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/Design

[26] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in action.* Manning, 2011.

[27] S. Schelter and S. Owen, "Collaborative filtering with Apache Mahout," *Proc. of ACM RecSys Challenge*, 2012.

[28] Welcome to Apache Pig! [Online]. Available: http://pig.apache.org/

[29] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* ACM, 2008, pp. 1099–1110.

[30] A. Gates, *Programming Pig.* "O'Reilly Media, Inc.", 2011.

[31] Solr. [Online]. Available: http://lucene.apache.org/solr/

[32] D. Smiley and D. E. Pugh, *Apache Solr 3 Enterprise Search Server.* Packt Publishing Ltd, 2011.

[33] A. Białecki, R. Muir, and G. Ingersoll, "Apache lucene 4," in *SIGIR 2012 workshop on open source information retrieval*, 2012, pp. 17–24.

[34] Overview of searching. [Online]. Available: https://cwiki.apache.org/confluence/display/solr/\Overview+of+Searching+in+Solr

[35] Spark. [Online]. Available: https://spark.apache.org

[36] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems.* ACM, 2013, p. 2.

[37] Graphx Programming Guide. [Online]. Available: https://spark.apache.org/docs/latest/graphx-programming-guide.html

[38] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell, *Learning Spark: Lightning-Fast Big Data Analytics.* O'Reilly Media, Incorporated, 2015.

[39] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012, pp. 10–10.

[40] Spark cluster image. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html

[41] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1013–1020.

[42] R. Kumar, N. Gupta, S. Charu, S. Bansal, and K. Yadav, "Comparison of sql with hiveql," *International Journal for Research in Technological Studies*, vol. 1, no. 9, pp. 2348–1439, 2014.

[43] M. Armbrust, "Unified data access with spark sql." Spark Summit 2014, San Francisco, 2014.

[44] T. Grainger and T. Potter, *Solr in Action*. Manning, 2014.

Appendix A

Apache License 2.0