ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

Department of Digital Systems

Digital Communications and Networks

Master Thesis

Study of technologies/research systems for big scientific data analytics

Surname/Name:	**Petsas Konstantinos**
Registration Number:	**ME 14039**
Supervisor:	**Rouskas Aggelos**
Delivery Date:	**30/09/2016**

Piraeus, 2016

Table of Contents

Table of figures

Abstract

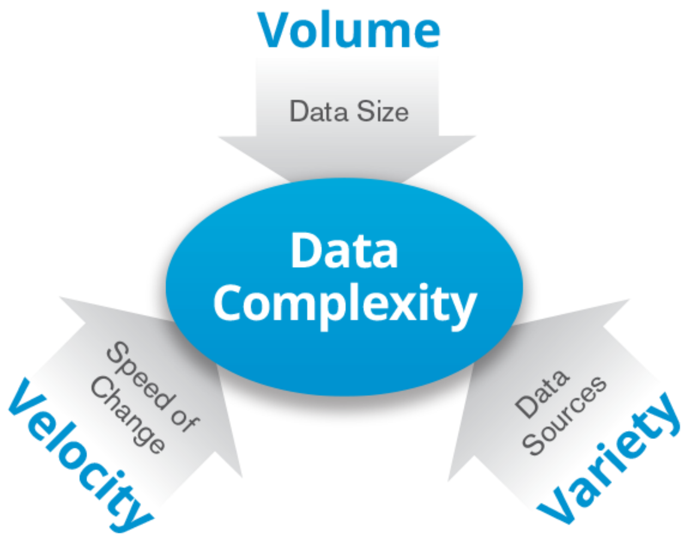
At this paper, are studied the existing technologies and the research systems, which are dedicated for the analysis of huge amounts of data, also known and often referred with the words “Big Data”. These large datasets could be, recorded data, or streaming data from Internet of Things devices that need to get analyzed with an upper purpose, such as the derivation of knowledge, in terms of learning a behavior.
At first, the focus is on the everyday rapid generation of data and how it can be managed by taking advantage of the Big Data analytics systems. The Apache Hadoop Java framework is the benchmark of these systems. Afterwards, there is a review of the existing technologies that are utilized and exploited for the processing of Big Data. The most of these technologies can integrate with the Hadoop framework and produce an enriched Big Data analytics ecosystem. In the next chapter, it is described the flexibility offered by a Hadoop cluster, in terms of adding nodes, in order to empower the distributed processing. Then, it is presented the installation procedure that should be followed, so as to create a Hadoop cluster, integrated with several components/technologies. Subsequently, there are a number of experiments that examine and evaluate the performance of a Hadoop cluster and some components of the Hadoop ecosystem, according to the number of the active data nodes. Finally, all the observation, conclusions and comments are concentrated in the last chapter, with thoughts for future exploitations of Big Data analytics systems.

1. Introduction

Every day, people generate millions of data only by using the social media websites. For instance, Facebook hosts billions of photos, with a growing rate of 7 petabytes per month (Source: The book *Hadoop: The Definitive Guide, 4th Edition*, by Tom White, April 2015, O'Reilly) [1]. Moreover, the Internet of Things (IoT) devices become more popular day by day and the increasing number of these devices means more data to process. More specifically, a set of IoT devices can produce tones of data in a day. The IoT sensors in some cases retrieve data uninterruptedly, with an upper purpose, such as the generation of knowledge, after a data processing activity. In order to be able, the data processing, to process large amounts of data, a Big Data Analysis is required as a pre-phase, in order to classify and filter the data. These facts were unpredictable, even by big companies and organizations before some years. The production of data nowadays has led to a reconfiguration of the infrastructure of many companies and organizations. The cost of the resources for the rescaling of the infrastructure is high enough. Another problem is that the scaling up has a physical limit, such as the size of the machine, the CPU, RAM, etc. [4]

The data being produced can be categorized according to three characteristics:

- Volume
- Variety
- Velocity



(Copyright 1995-2015 GRT Corporation)

Figure 1.1: The three V's of Big Data. [5]

Volume is the main concept of big data. It is the large amount of data to be stored and analysed. Velocity is the execution time, in terms of the time required to access the data and find something specific, in a dataset that consists of exabytes or even more. Variety is about the inconsistency of the data. To analyse, the data generated nowadays, can be from various sources, heterogeneous, so the most of the data to be stored and processed is unstructured. The Big Data gives solutions to these three V's mentioned above, by managing and analysing such datasets. [4], [5]

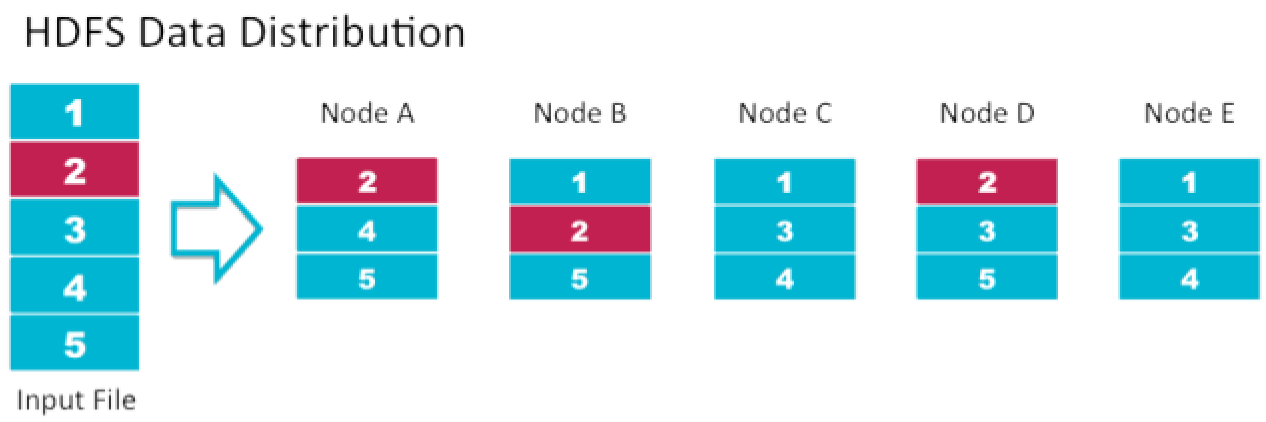
1.1 Big Data

The three V's mentioned above gave the trigger for the creation of Big Data tools and frameworks. The target is to handle situations that older systems cannot. The capabilities of the Big Data tools and frameworks are empowered by some specific characteristics.

1. Firstly, the **data distribution** is about the division of the data to smaller blocks and the distribution of these blocks to the available nodes of the cluster.

2. The data exists in the Distributed FileSystem (DFS) and after the distribution is ready for **parallel processing**. Every node of the cluster is a powerful server which can analyse the block of data residing in it. Every node processes the data simultaneously, in order to classify and filter the data to achieve the required result.

3. **Fault tolerance** is another important characteristic. Every block of data is being kept in many nodes, as a replica, so as to ensure that the data is available at any time, even if a node server is down for any reason.



(Copyright 2013, BigData Planet)

Figure 1.1.1: Replication of data blocks [4]

4. Big Data tools and frameworks are economy efficient as they only need **common hardware** in order to operate, with no special demands.

5. Last but not least, concerns the **flexibility** and the **scalability** they offer. The cluster architecture is offered for addition of nodes in order to increase the space and the process power and efficiency.

Regarding all these characteristics, complex problems, such as the three V's can be eliminated. [4]

1.2 Document Organization

This master thesis consists of the following chapters. There is also, a small description for every chapter.

Chapter 2: The Apache Hadoop framework will be analysed, in terms of architecture and the mechanisms included for the classification of the big data, through a built-in example, explaining also the Map-Reduce procedure.

Chapter 3: There will be a theoretical reference to components/technologies that can be combined with the Hadoop and offer an enriched ecosystem for the better data analysis, storage and workflow creation. In order to create workflows we use the Apache Oozie, which is a workflow scheduler for managing Hadoop jobs. We will use Hue as a Web interface for the creation of a workflow, that will manage Hadoop jobs.

Chapter 4: In this chapter it is described the procedure of scaling up the Hadoop Cluster and what are the benefits of this action. Also, the removal of a Node from the cluster will be described.

Chapter 5: This chapter is dedicated to the installation procedure that should be followed in order to create a Hadoop cluster, setup Apache Pig and Oozie components and install Hue as a web user interface for the Hadoop Distributed filesystem, which also offers editors for the components of the Hadoop ecosystem.

Chapter 6: In this chapter there are some experiments, in terms of running workflows with oozie job scheduler. The jobs will be Pig scripts that will analyse a large file in a specific approach, defined in the scripts. The same experiments will run in a four node cluster and then, in a three node cluster, so as to evaluate the results and compare the required time to run a workflow before the scaling of the Hadoop cluster and after the scaling on demand. To prove the benefits of scaling up a Hadoop Cluster on demand, there is also a Map Reduce job that will run on a file with a million lines.

Chapter 7: Finally, there is a report of the conclusions conducted during the writing of this master thesis and through the implementation and testing of the experiment.

2. Apache Hadoop framework

2.1 What is Hadoop?

Hadoop is an open source framework, implemented in Java, which offers parallel and distributed data processing. It is the most commonly used framework in the industry for the analysis of big datasets. The first releases of Hadoop, Hadoop 1.X, started with two main components, the Hadoop Distributed FileSystem (HDFS) and the MapReduce. There were some drawbacks by using Hadoop 1.X.

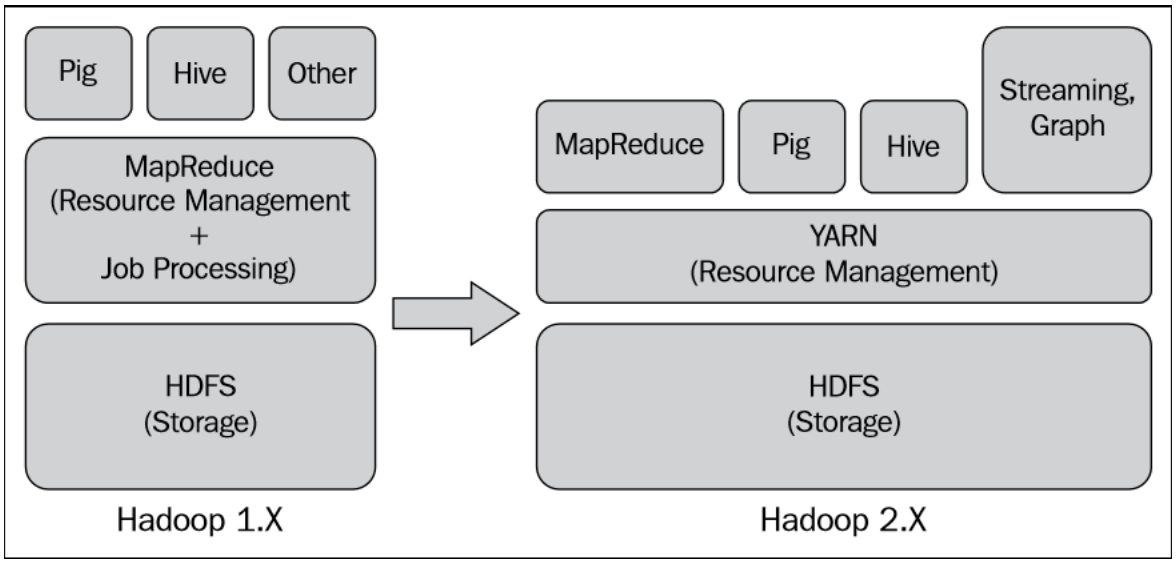
For example, there was only one, centralized, JobTracker, which had to perform many activities like Resource Management and Allocation, Job scheduling and execution etc. In case of a failure, the jobs in the system had to start all over again. Also, there was only one Name Node that stored all the metadata about the files/blocks stored in the HDFS. In addition, Hadoop 1.X was mainly Unix based and big companies running Windows Microsoft servers could not use Hadoop. [2]

In order to overcome the limitations of Hadoop 1, the architecture changed by adding a new component. The upgrade to Hadoop 2 led to a more flexible system, in terms of availability and scalability. Hadoop 2 is characterized and always followed by the abbreviation YARN which will be explained in the next subsection.

2.1.1 Yet Another Resource Negotiator (YARN)

In Hadoop 1.X the main supported model was MapReduce. The adoption of Hadoop by the enterprise led to the need of Hadoop to offer more than MapReduce features. The initial target of YARN was to separate the resource management from the job/application execution. In this way, more applications could be added to a Hadoop computing cluster. The architecture of YARN abstracts out the **Resource Manager**, which is responsible for the monitoring of the resource usage in the cluster, the activities taking place and also, allocates the resources. The planning and the execution of Hadoop jobs are handled by the **Application Master**. Every application has its own Application Master. In the YARN architecture the MapReduce is considered as an application, so if there are three

MapReduce jobs running, every one of them will have their own Application Master. [2]



(Copyright 2014, Sandeep Karanth, Mastering Hadoop)

Figure 2.1.1.1: Difference of architecture between Hadoop 1.X and 2.X. [2]

In order to run any application (e.g. MapReduce, Pig, Hive) the Application Master will request resources from the Resource Manager, in terms of RAM, CPU and memory.

In every node of the Hadoop cluster there is a daemon running, called **Node Manager**. The Node Manager is the contact point between the Resource Manager and the available nodes. The jobs to run are scheduled by the Resource Manager, keeping metadata in order to recover in case of any Resource Manager crash. The Application Master then, requests resources, executes the tasks and handles any job failures. [2]

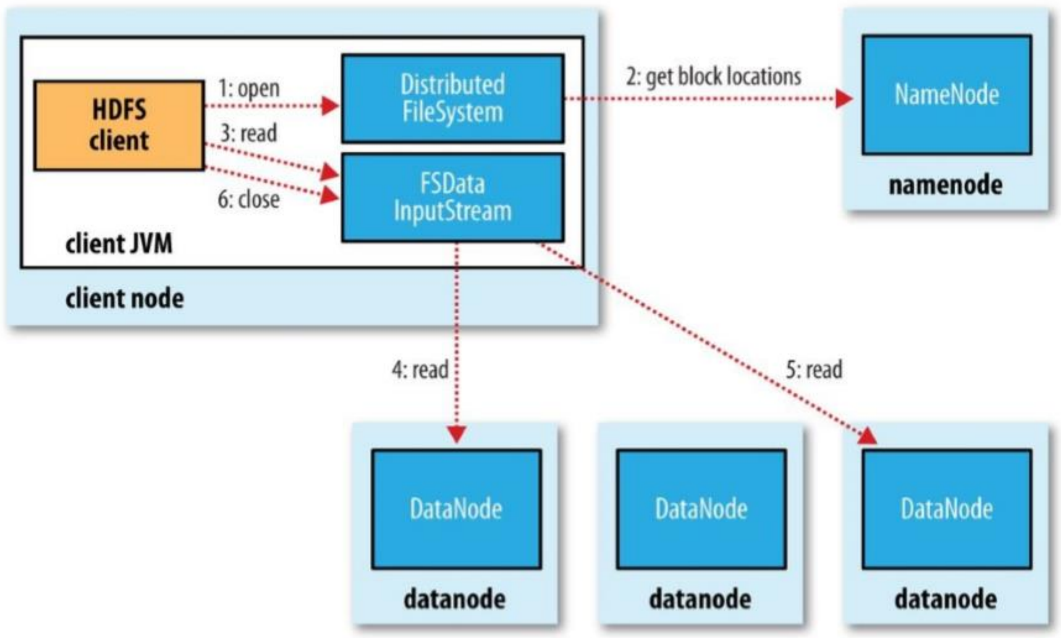
So the new processing models in Hadoop are offered by YARN in Hadoop 2. It consists of a cluster with a resource management system, allowing every distributed program to run and analyse big datasets in a Hadoop cluster.

2.2 Hadoop Distributed File System (HDFS)

The HDFS is a filesystem, composed by the machines of a cluster that run in commodity hardware, dedicated for the storage of very large files, such as petabytes of data. HDFS also offers a streaming process to access the data, which is useful for the reading of such large datasets. When a large file is saved in HDFS, it first breaks in block-sized chunks that finally stored as independent units. The default block size in HDFS is 128 MB, which is a respectful amount of data comparing to the 512 bytes stored in disks. The block abstraction is useful in a distributed filesystem because this means that a single file can be larger than the disk of any machine in the network. Moreover, it is easy to manage storage subsystem, as the blocks are fixed sized and with simple math a calculation can show how many blocks can be stored in the given disks. Also, the metadata holding e.g. permission information can be stored separately from the blocks, in another system. Furthermore, blocks offer fault tolerance and availability at any time, as they replicated in more than one nodes. The replication factor can be set by the owner of the cluster. So when a client asks for a specific block, if this block is corrupted, it can be retrieved by an alternative location, seamlessly to the client and re-replicated so as to keep the replication factor to the normal level. [1]

2.2.1 Hadoop Name Node and Data Nodes

A Hadoop cluster has two types of nodes. The Name Node, known as master and the Data Nodes, known as slaves – workers. The namespace of the filesystem is managed by the Name Node. This node is aware of the filesystem tree and the metadata of every data in the tree. Every Data Node is registered in the Name Node and also knows where to request the blocks of a given file. Once the system is up and running, the Data Nodes report to the Name Node which blocks are stored in their disks. Also, Data Nodes store or retrieve blocks on Name Node’s demand.



(Copyright 2015, Tom White, Hadoop: The Definitive Guide, 4th Edition)

Figure 2.2.1: HDFS client reads data from HDFS [1]

The Name Node is the most important node, because without it the filesystem is useless. The files can be found only through this node, otherwise they are lost and the reconstruction of a file from the blocks is impossible. This is extremely dangerous so the Hadoop provides two solutions to overcome this risk.

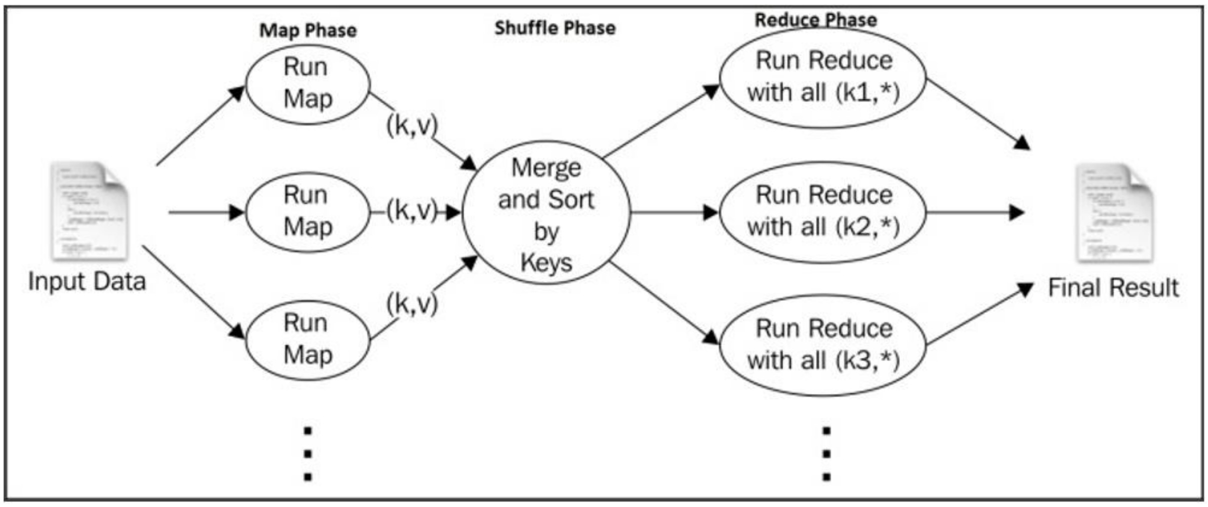
The first one is to write the filesystem metadata persistently to the Name Node and synchronize the persistent state with the local disk or a remote NFS mount.

The second solution is to have a secondary Name Node, which will not work as a Name Node if not needed. The secondary Name Node runs in a separate physical machine and is available in case of a Name Node failure. [1]

2.3 MapReduce

MapReduce is a data processing programming model, which can process large amounts of data, stored in HDFS. Through the MapReduce procedure there are two functions taking place. The first phase is the Map function and the second the Reduce. [6]

The Map function receives as input data lines of a file, rows of a database and so on, as key-value pairs and produces a result in the same way. A single Map task instance is created for every data block in the HDFS, which is relevant to the input data. The Map functions inside a Map task instance are equal to the number of data records of the input block. The computation is executed in parallel. After the mapping procedure, the output data is grouped and sorted by key and is ready to get as input data to the Reduce function. This phase of handing over the data from map function to reduce function is called *Shuffle Phase*. The reduce function iterates through the key-value pair list, so as to implement summarization operations on data. [1], [2], [6]



(Copyright 2015, Thilina Gunarathne, Srinath Perera, Hadoop MapReduce v2 Cookbook, Second Edition)

Figure 2.3.1: MapReduce procedure phases. [6]

In Hadoop 1.X the MapReduce components were the JobTracker and the TaskTrackers. The first one was running on the Name Node and was responsible for the cluster management and the jobs coordination. The other one was running in every Data Node and it was the task launcher and coordinator, for the tasks being executed to each node. These processes no longer exist in the Hadoop 2.X. The Application Master coordinates the jobs and the cluster management and job scheduling is handled by the YARN Resource Manager. There is also a JobHistoryServer which provides information about the completed jobs.

Through the above statements we can understand that the Hadoop 2 has changed the way that the MapReduce application runs on a cluster. In order to keep compatibility with MapReduce applications written in Hadoop 1, the YARN team has written a MapReduce framework, which runs on top of YARN. In this way, companies that have wrote MapReduce algorithms in Hadoop 1, through a simple procedure can reproduce these algorithms to Hadoop 2. It only needs a recompilation of the existing program. [3], [6]

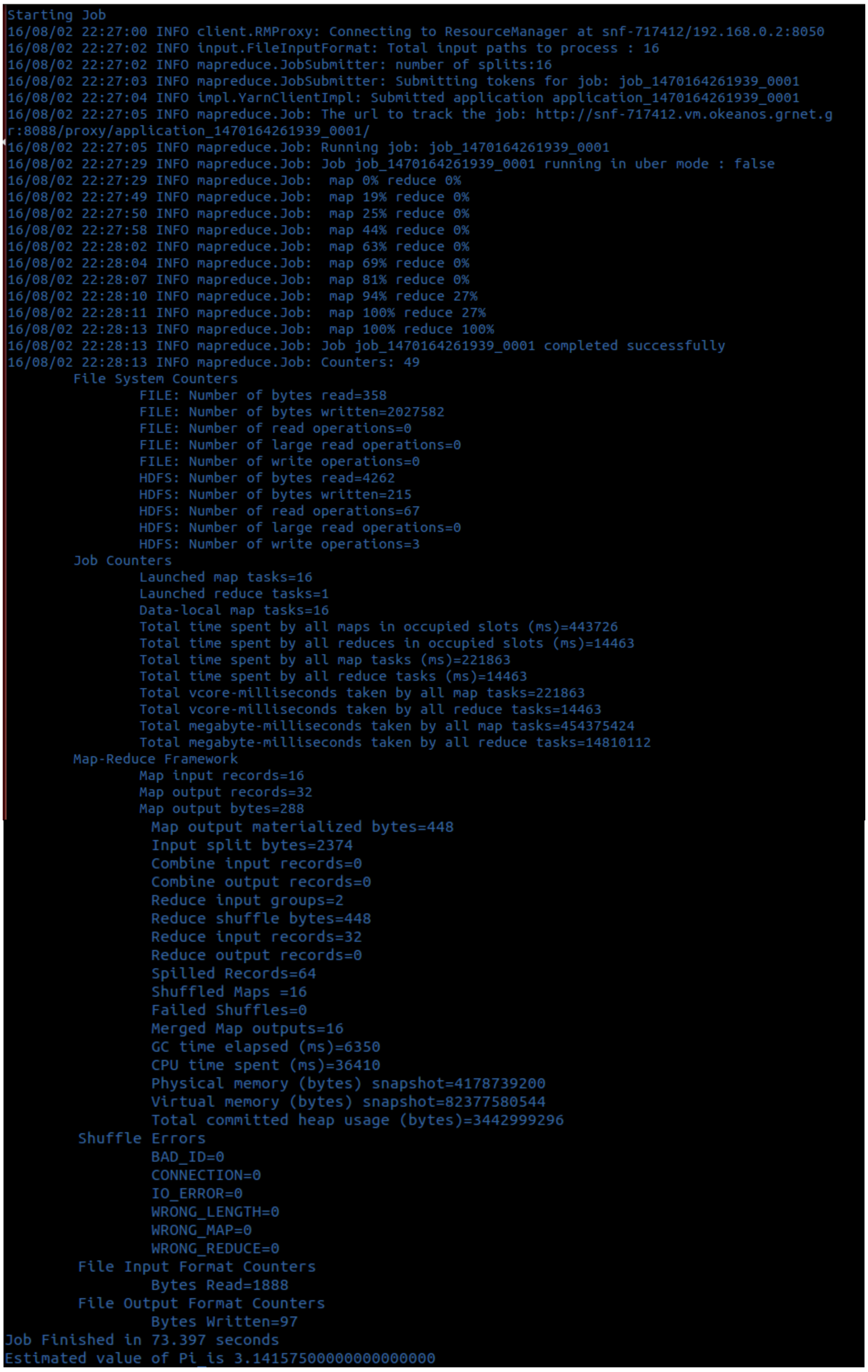
2.3.1 YARN MapReduce Examples

After the installation of Hadoop, there are available some MapReduce examples. In our current Hadoop installation the MapReduce examples are in the folder path /usr/local/hadoop/share/hadoop/mapreduce. The most common example is the WordCount.

The WordCount example takes as input a file with text and produces an output file which has a list with every word that exists in the whole text followed by a number that shows the frequency of appearance of every word. The procedure starts by splitting the file to different parts. Then these parts will be distributed in the Data Nodes and will be parallel processed by the Map function. The output of the Map procedure will be a file with the words as a key and the frequency of appearance as a value. These key-value pairs, from every Data Node, will be the input to the Reduce function which will add the frequency of appearance for the words that are the same, coming from different Map functions. The final output will be a file with the words and their frequency of the appearance in the text given as input.

Another example is the Pi that computes the value of mathematical pi. By running the following command in the command line, the pi is calculated with 16 map functions and 100,000 samples. Finally, we take the result shown in the Figure below.

```
$ yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar pi 16 100000
```



```
Starting Job
16/08/02 22:27:00 INFO client.RMProxy: Connecting to ResourceManager at snf-717412/192.168.0.2:8050
16/08/02 22:27:02 INFO input.FileInputFormat: Total input paths to process : 16
16/08/02 22:27:02 INFO mapreduce.JobSubmitter: number of splits:16
16/08/02 22:27:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1470164261939_0001
16/08/02 22:27:04 INFO impl.YarnClientImpl: Submitted application application_1470164261939_0001
16/08/02 22:27:05 INFO mapreduce.Job: The url to track the job: http://snf-717412.vm.okeanos.grnet.g
r:8088/proxy/application_1470164261939_0001/
16/08/02 22:27:05 INFO mapreduce.Job: Running job: job_1470164261939_0001
16/08/02 22:27:29 INFO mapreduce.Job: Job job_1470164261939_0001 running in uber mode : false
16/08/02 22:27:29 INFO mapreduce.Job:  map 0% reduce 0%
16/08/02 22:27:49 INFO mapreduce.Job:  map 19% reduce 0%
16/08/02 22:27:50 INFO mapreduce.Job:  map 25% reduce 0%
16/08/02 22:27:58 INFO mapreduce.Job:  map 44% reduce 0%
16/08/02 22:28:02 INFO mapreduce.Job:  map 63% reduce 0%
16/08/02 22:28:04 INFO mapreduce.Job:  map 69% reduce 0%
16/08/02 22:28:07 INFO mapreduce.Job:  map 81% reduce 0%
16/08/02 22:28:10 INFO mapreduce.Job:  map 94% reduce 27%
16/08/02 22:28:11 INFO mapreduce.Job:  map 100% reduce 27%
16/08/02 22:28:13 INFO mapreduce.Job:  map 100% reduce 100%
16/08/02 22:28:13 INFO mapreduce.Job: Job job_1470164261939_0001 completed successfully
16/08/02 22:28:13 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=358
                FILE: Number of bytes written=2027582
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=4262
                HDFS: Number of bytes written=215
                HDFS: Number of read operations=67
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
        Job Counters
                Launched map tasks=16
                Launched reduce tasks=1
                Data-local map tasks=16
                Total time spent by all maps in occupied slots (ms)=443726
                Total time spent by all reduces in occupied slots (ms)=14463
                Total time spent by all map tasks (ms)=221863
                Total time spent by all reduce tasks (ms)=14463
                Total vcore-milliseconds taken by all map tasks=221863
                Total vcore-milliseconds taken by all reduce tasks=14463
                Total megabyte-milliseconds taken by all map tasks=454375424
                Total megabyte-milliseconds taken by all reduce tasks=14810112
        Map-Reduce Framework
                Map input records=16
                Map output records=32
                Map output bytes=288
                Map output materialized bytes=448
                Input split bytes=2374
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=448
                Reduce input records=32
                Reduce output records=0
                Spilled Records=64
                Shuffled Maps =16
                Failed Shuffles=0
                Merged Map outputs=16
                GC time elapsed (ms)=6350
                CPU time spent (ms)=36410
                Physical memory (bytes) snapshot=4178739200
                Virtual memory (bytes) snapshot=82377580544
                Total committed heap usage (bytes)=3442999296
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1888
        File Output Format Counters
                Bytes Written=97
Job Finished in 73.397 seconds
Estimated value of Pi is 3.14157500000000000000
```

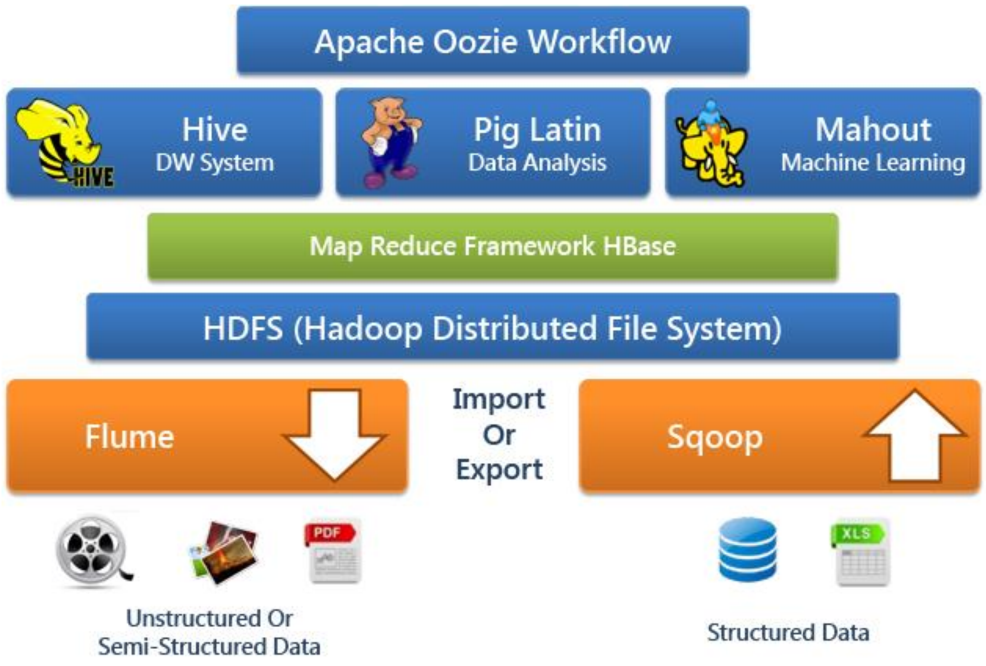
Figure 2.3.1.1: Pi MapReduce example.

In order to run the Pi MapReduce example and produce the above result, I created a Hadoop Cluster with three nodes, one Name Node and two Data Nodes, in the ~Okeanos cloud, which offers the Infrastructure as a Service (IaaS) for creating Virtual Machines (VMs), that are used for the formation of a Hadoop Cluster and a HDFS where the example/job runs. The executed command is based on an example from the [3] book reference.

3. Hadoop Ecosystem

Hadoop is a framework for managing and analysing Big Data, but it also offers more than that. There are many technologies that can be combined with the Hadoop framework, for the processing of different types of data. All these technologies were built in order to offer a powerful Big Data Platform, which can be used by big corporations for many cases, such as the creation of a database table with billions of rows and millions of columns. When the Hadoop is combined with these technologies, the result is a Hadoop Ecosystem. [7]

The components of the ecosystem can be Apache Hive, Apache Pig, Apache Oozie, Apache HBase and many more. In order to create the ecosystem, it is necessary to create a Hadoop installation first. Then, one by one the components will be added, by configuring the cluster to accept the changes.



(Copyright 2016, Stratapps Inc.)

Figure 3.1: Hadoop Ecosystem. [8]

As depicted in the above figure, there are three types of data under the Big Data Platform. **Structured data** is the easiest to be handled data, as it has a proper structure and can be stored traditionally in relational databases, such as MySQL. [7]

Semi-Structured data has some structure, but it is not possible to be saved in relational

databases because the table format is not suitable for such data. For instance, XML data or email messages will not be stored in a table.
Unstructured data does not have any structure at all. Relational databases are out of the question. An example of this data is a video file. [7]

Below, the components of the ecosystem displayed in the figure, will be analysed. The Hadoop core components, Hadoop Distributed FileSystem (HDFS) and MapReduce, are studied in previous sections, so the focus is on the rest technologies.

3.1 Apache Hive

In order to use Hadoop, companies needed to have expert Java developers specialized in MapReduce coding, even for simple tasks. This led to the creation of Hive, which would be accessible to a wider set of developers. As many programmers are familiar with Structured Query Language, Hive provides an interface similar to the common SQL interfaces. This is why Hive is also known as Hive Query Language or HiveQL. Hive queries are used for the extraction of data out of the Hadoop system. [7], [9]
Hive works as a converter, from SQL queries, to a series of MapReduce jobs for execution on a Hadoop Cluster. The data is organized in tables and by this way, Hive offers some kind of structure to the data stored in HDFS. These table schemas are the Metadata, which are stored in a database called The Metastore. [1]
Due to the conversion of the SQL queries to MapReduce jobs, Hive has a higher latency comparing to the traditional databases, because of the start-up overhead. But Hive is meant to be used for the processing of jobs on huge immutable data, such as Application Logs. [9]

Hive is a distributed data warehouse, design for easy data aggregation, ad-hoc querying and analysis of large amount of data. Hive also provides some services that will be discussed in the next subsection. [9]

3.1.1 Hive Services

The **hive shell** is the primary way of interaction with Hive, by writing commands in HiveQL, which is a query language similar to MySQL. [1]
For instance, in order to list the available tables the proper command is:

```
hive> SHOW TABLES;
OK
Time taken: 0.473 seconds
```

The first time that the Hive command runs, it takes a few more seconds, because it creates also The Metastore database. To access the hive shell, the user only runs the command hive. [1]

Except for the above service there are also some more useful hive services.

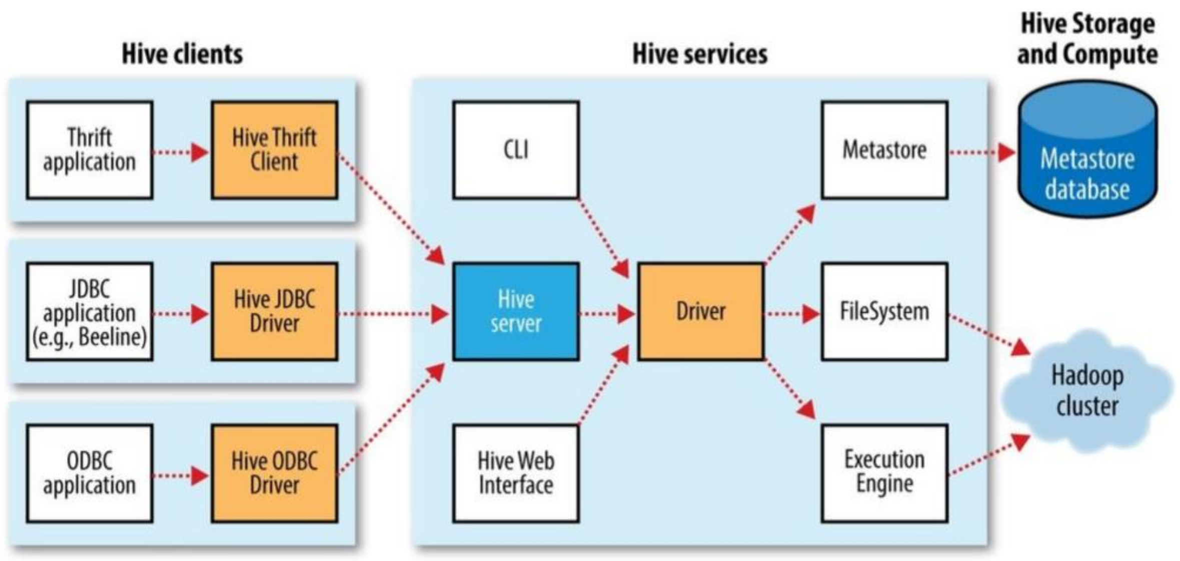
The **Hiveserver2** runs Hive as a server supporting authentication and multi-user concurrency. It is accessible by clients written in many different languages. It uses the Thrift service, which is an interface definition language, to bridge communication between the Hive server and Hive. [1]

Another Hive command line interface is the **beeline**. It is available in embedded mode or through a connection with a Hiveserver2 using JDBC (Java Database Connectivity), which is an application programming interface (API) for the interconnection of the programming language Java with an SQL database. [1]

There is also the possibility to access Hive through a simple web interface. **HWI** (**Hive Web Interface**) is an alternative to the CLI, without having to install any client software. In a following section is presented also Hue, which is a Hadoop web interface for the whole ecosystem, including Hive. It offers applications for running hive queries and browsing the hive Metastore. [1]

The hive also provides the command jar, equivalent to the Hadoop command `hadoop jar`, which runs Java applications with Hadoop and Hive classes on the classpath.

Finally, the **Metastore** runs in the same process as the Hive service. The Metastore can also run as a standalone process, as a server and the user can set the port to listen on. [1]



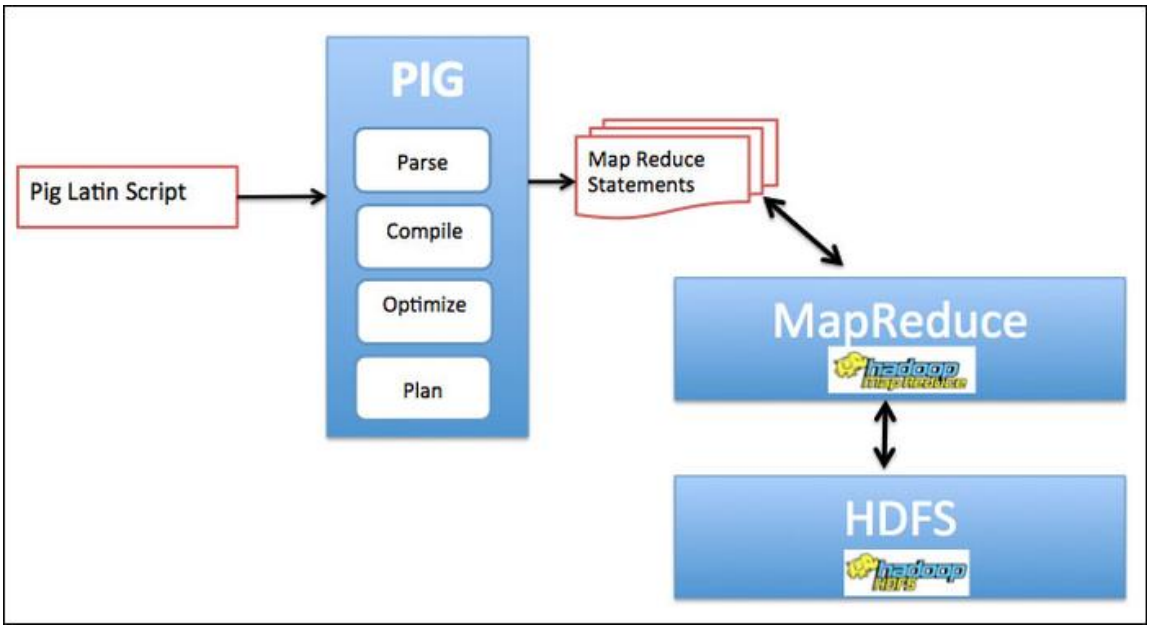
(Copyright 2015, Tom White, Hadoop: The Definitive Guide, 4th Edition)

Figure 3.1.1.1: Hive Architecture. [1]

The central repository of Hive metadata is The Metastore. By default, the Metastore service runs on the same JVM (Java Virtual Machine) as the Hive service and contains an embedded Derby database, which does not support multiple Hive sessions open at the same time accessing the same Metastore. Of course, by applying no complex configurations everything can change and multi user sessions on the same Metastore can be supported. [1]

3.2 Apache Pig

Pig is a high-level platform for developing programs, in Pig Latin language, that run on Hadoop. Pig Latin is a data flow programming language, which has an execution environment and runs on HDFS and MapReduce Clusters. It is used for the exploration of very large datasets. It is similar to Hive, in terms of dealing with structured data. Pig was developed at Yahoo for the same reason as Hive. It provides an alternative to developers who prefer scripting languages, such as Python, to interact with the Hadoop. A pig program applies a series of operations and transformations to the input data, which in the backend runs MapReduce processes to produce the desired result. The Pig after the script execution, it prepares a series of MapReduce jobs, which in local mode run on JVM and in MapReduce mode run on the Hadoop Cluster. [1], [7], [8]



(Copyright 2013, Rajesh Nadipalli, HDInsight Essentials)

Figure 3.2.1: Pig Architecture. [10]

The Pig command line is called Grunt. A statement is considered to be an operation or a command. A Pig Latin program consists of many statements. These statements are parsed by the Pig platform. The above figure shows that the Pig Latin Scripts or the Pig commands inserted and executed in the Grunt shell will be parsed by the Pig platform. After the parsing, the Pig will compile, optimize and fire MapReduce statements. Finally, MapReduce accesses HDFS and returns the results. [10]

3.3 Apache Oozie

« Oozie is a workflow scheduler system to manage Apache Hadoop jobs ».

(Source: the official Apache Oozie website, Last Published: 2016-08-17, Online available: http://oozie.apache.org/, Accessed at 19-Sept-2016) [22].

It is a server-based system for running workflows of dependent jobs. It is composed of a workflow engine and a coordinator engine. The first one stores and runs workflows composed of different types of Hadoop jobs, such as MapReduce, Pig, Hive and so on. This proves that oozie is fully integrated with the rest of the Hadoop ecosystem. The coordinator engine runs workflow jobs based on predefined schedules and data availability. [1]
Oozie has high scalability and can execute a large number of workflows in a Hadoop Cluster, even if each workflow is composed of many jobs dependent to each other. [1]
Oozie is implemented as a Java web-application (Source: Wikipedia , the free encyclopedia, Online available: https://en.wikipedia.org/wiki/Apache_Oozie, Accessed at 19-Sept-2016) [23] and it is useful when a big data programmer needs to use the output of a Hadoop job A, as input to a Hadoop job B and the output of job B as input to a job C and so on. To automate this sequence of jobs, Apache Oozie is utilized. [7]

Oozie comes with the benefit of rerunning failed workflows, which is important in order to get the desired result. It runs as a service in the cluster and clients can submit their workflow requests for instant or later execution. After the workflows are completed, Oozie informs the client through an HTTP call back about the workflow status. [1]

3.3.1 Workflows

In order to define an oozie workflow, the programmer has to write an XML (Extensible Mark-up Language) according to the Hadoop Process Definition Language, which is specified on the Oozie Website. A workflow is a series of actions encoded by XML nodes. Some nodes are responsible for the execution if some actions and others are responsible for the flow control, according to what has been defined in the XML file. Every workflow node has a unique identifier, because every node has a specific action to execute and the order they appear in the XML is important. The below XML file shows an oozie workflow. [11]

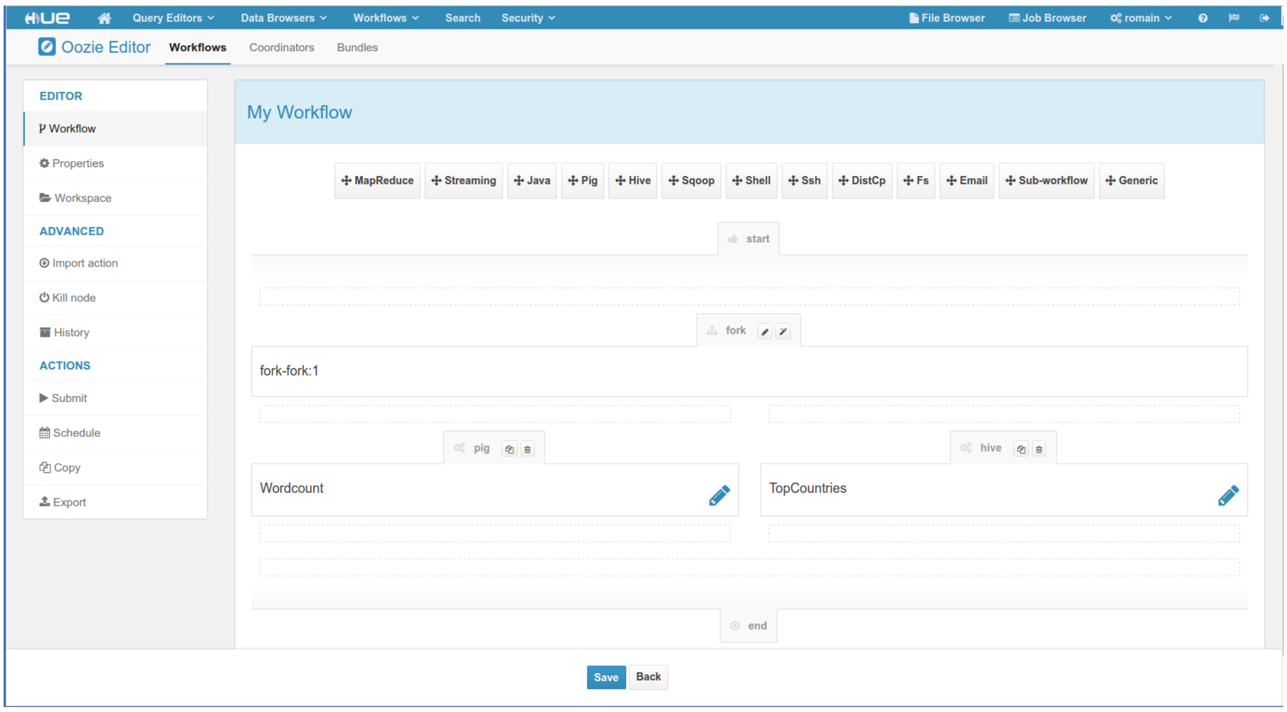
```
<workflow-app name="SampleWorkflow" xmlns="uri:oozie:workflow:0.1">
  <start to="firstJob"/>
  <action name="firstJob">
    <pig>...</pig>
    <ok to="secondJob"/>
    <error to="kill"/>
  </action>
  <action name="secondJob">
    <map-reduce>...</map-reduce>
    <ok to="end" />
    <error to="kill" />
  </action>
  <end name="end"/>
  <kill name="kill">
    <message>"Killed job."</message>
  </kill>
</workflow-app> [11]
```

In the above workflow example there are three control-flow nodes, handling the start, end and kill and two action nodes which represent the execution of an application or a command. [11]

As the above XML depicts, the first node is a control-flow node that redirects the workflow to the first action node, which will run a Pig script or command. If the first job completes with no errors then the workflow will continue to the second action node for the execution of a MapReduce application. Finally, if the second job is successful, the workflow will continue to the end control-flow node and will report that the whole workflow job was successful. In case any of the action nodes fails to execute the corresponding job, the workflow will transition to the kill node, which will report back the specified error message.

It is necessary for all workflows to have one start and one end node. The initial step of a workflow job is the transition to the node that is specified by its unique identifier in the start section in the XML. A workflow job is considered successful when it reaches the end node. In case a workflow job transitions to a kill node, it automatically means that the workflow job has failed and the error message specified in the corresponding XML section is reported back to the user/programmer. [1]

Another way of creating workflows is with Hue, which is a Hadoop Web Interface and will be described in a following section. Hue provides a graphical user interface where the user/programmer can easily drag and drop jobs, for example a pig script, in the workflow editor. After the workflow job creation there is the possibility to execute it or even save it in order to re-execute it another time. The Oozie workflow editor offered by Hue is depicted in the figure below.



(Copyright 2015, Hue Team, gethue.com)

Figure 3.3.1: Hue – Oozie workflow editor. [12]

The figure depicts a workflow job with two control-flow nodes, start and end and three action nodes. The action nodes consist of a fork job as the initial action node, which means that the start node will redirect the workflow to the fork node and after the node's success the workflow will transition to the next action node, which consists of a workflow and a sub-workflow. The main workflow is the execution of the Pig script and the WordCount job and the sub-workflow is the execution of the Hive query for the Top Countries. The main workflow will wait for the sub-workflow to be completed, so as to continue and use the output produced. Finally, the workflow job will reach the end node and will report the status of the procedure.

3.4 Apache Mahout

Apache Mahout is an open source Machine Learning library, for scalable algorithms, written in Java. It is implemented on top of Hadoop and uses the MapReduce. The algorithms it offers are machine learning or collective intelligence. The main purpose of Mahout is to provide to developers a machine learning tool, which has the characteristics of filtering,

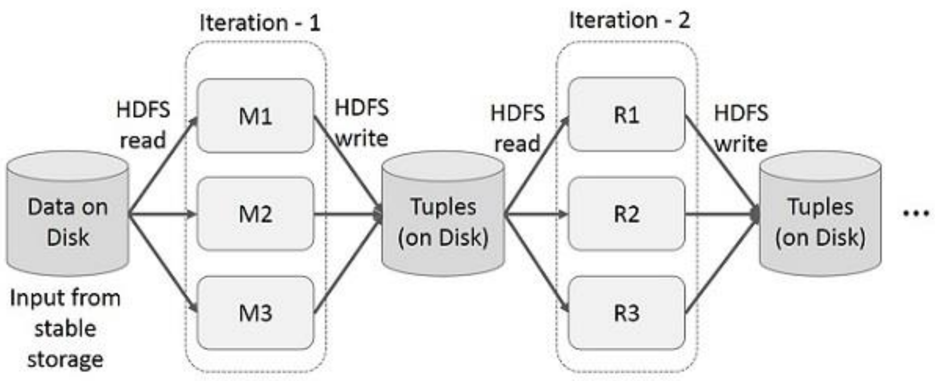
clustering and classification, for the cases where the data to be processed is so huge, that cannot be processed by a single machine. [7], [8]
Also it provides a programming environment and framework for the creation of scalable algorithms. [7], [8]

3.5 Apache Spark

Apache Spark is a cluster computing framework that offers fast data processing and analysis on large datasets. Although spark has built on HDFS, it does not use MapReduce as an execution engine but it uses its own distributed data processing framework. [1], [13]
The strong characteristic of spark is the ability of keeping huge working datasets in memory between running jobs. This benefit makes spark to have a better performance than MapReduce workflows, which needs to load every time the datasets from the disk. Spark is most suitable for applications that run iterative algorithms, where there is a function being applied repeatedly to a large dataset, and applications for interactive analysis, where a user creates many ad hoc queries for the exploration of a dataset. [1]
Apart from the in-memory caching, that spark offers, it also is a Directed Acyclic Graph (DAG) engine. As a result, it can process numerous pipelines of operators and translate them into a single job for the user, something that MapReduce is not able to achieve. [1]
These special characteristics make Apache Spark to be a suitable platform for applications which include real-time queries, as it provides the Spark SQL module for SQL operations, machine learning with the MLlib module, event stream processing with the Spark Streaming module, graph processing with the GraphX and complex operations. [1], [13]
Also, Spark provides to the developer/user an enriched set of APIs for the execution of many common data processing tasks and APIs for the creation of programs in three different languages: Scala, Java and Python. Scala is a general-purpose programming language. [1]

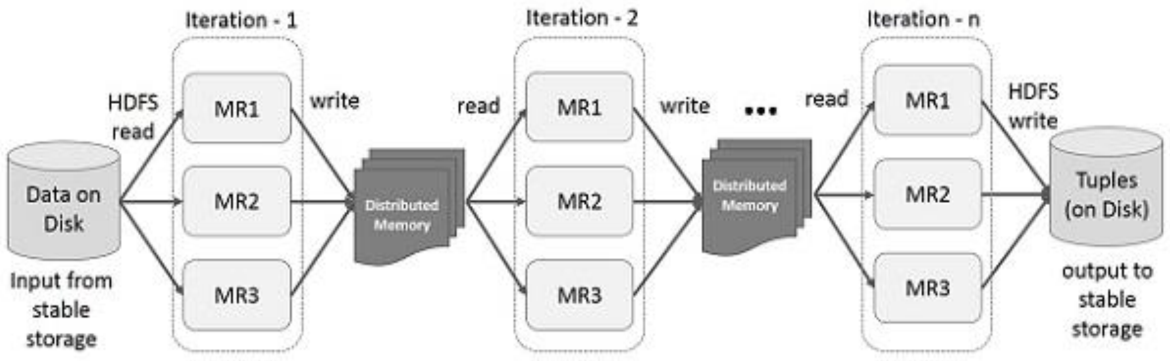
In order to run a job, which in this case is made of *an arbitrary directed acyclic graph (DAG) of stages*, Spark splits these stages into tasks that run in parallel across the cluster. Spark is based on the concept of RDDs (Resilient Distributed Datasets). The processes running for the execution of a job are handled as they are a local collection of operations. RDDs are operating in parallel on a Hadoop cluster and handled the same way as Scala treat sequences. RDDs are fault tolerant, but if an RDD operation fails, it has to start all over again. There are two RDD categories: the transformations and the actions. Transformations create new RDDs, such as map(), filter(), groupBy() and they are executed only when needed. The actions include the result returning and the output saving to the storage system. The functions included are for example, collect(), count() and save(). [14], [1]

Below figures show the different processing of data of Spark and MapReduce without Spark.



(Copyright 2016, tutorialspoint.com)

Figure 3.5.1: Map Reduce procedure without Spark. [15]



(Copyright 2016, tutorialspoint.com)

Figure 3.5.2: Map Reduce procedure with Spark RDD. [15]

In the above figures the in-memory caching concept of Spark is clarified. The exploitation of Spark, make the system clearly faster, due to the distributed memory where the results of the iterations are saved. [15]

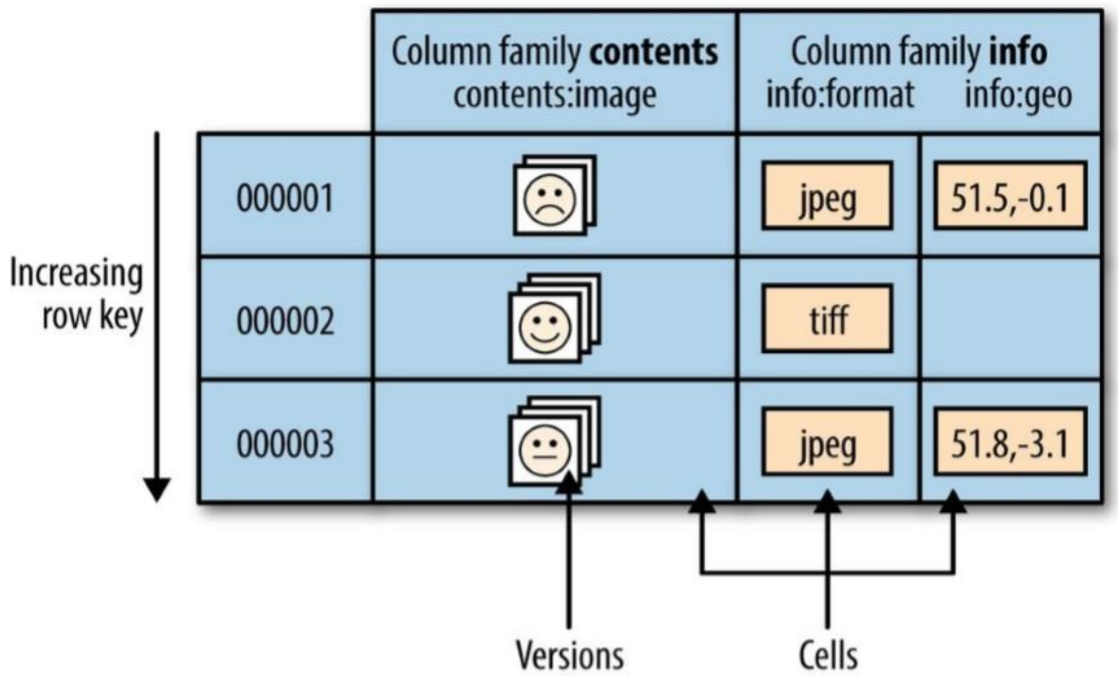
In order to integrate the Apache Spark with the rest of the Hadoop components, it should run on YARN of an existing Hadoop Cluster. Spark can be deployed on YARN through the *YARN client* mode or the *YARN cluster* mode. In the first case the driver runs on the client and in the second one, on the YARN application Master. [1]

YARN client mode is most suitable when using interactive components, such as spark-shell in the console and pyspark, or when writing spark programs. [1]

YARN cluster mode is dedicated in production jobs. The reason is that the application runs completely on the cluster and it is fault tolerant, as the YARN will retry the execution of an application if the application master fails. [1]

3.6 Apache HBase

HBase is an online key – value store built on top of HDFS. It is a column-oriented database that uses the HDFS as its underlying storage. [1]
HBase is the most suitable component of the Hadoop ecosystem for applications that require real-time read/write operations with random access to huge amounts of data. It is multidimensional and in order to get a value, the user should provide a combination of a row key and a column key. It also supports millions of columns per row. The stored values are under a version, so when a value is replaced, it is not really deleted but a new version entry is being created. [1], [14]
Considering the HBase table row keys, they are byte arrays, so any type of data can be a row key, from strings to binaries. [1], [14]



(Copyright 2015, Tom White, Hadoop: The Definitive Guide, 4th Edition)

Figure 3.6.1: The HBase Data Model. [1]

The columns of a row are grouped and they called column families. Every column family has a specific prefix. The family members of a column have a common prefix, as shown in the figure, such as the columns “info:format” and “info:geo”. Moreover, the sorting of the table rows are according to a primary key. [1]

HBase is categorized in NoSQL (Not Only SQL) Databases. More information about NoSQL databases are presented below, by comparing the Relational Data-Base Management Systems (RDBMSs), such as MySQL, with the HBase NoSQL database.

The main characteristics of a **RDBMS** are that it is **schema-oriented**, it has **normalized data** and it contains **thin tables**. [14]

- Concerning the first characteristic, it means that the tables have a fixed schema, with a predefined type of data to be saved. The definition takes place during the creation of the table. The data to be inserted in the table is highly structured data. [14]
- RDBMSs store normalized data, but when they work as data warehouses the data can be de-normalized. [14]

- RDBMSs have a maximum number of columns that is limited to hundreds of columns. This results in the creation of multiple tables, which have different relationships with its other. More specifically, one-to-one, one-to-many and many-to-many. [14]

In contrast, the NoSQL databases, therefore HBase, are **schema-less** and they can handle **de-normalized type of data**. [14]

- The schema-less characteristic is achieved through the mapping of the data. More specifically, the columns can be defined at runtime and every row has its own columns. The application handles the interpretation of the values to be stored or retrieved from the HBase or any other NoSQL database. This offers a flexible schema which is useful in many cases where the data to be saved varies in many ways. [14]
- The de-normalized type of data offers, for example, the possibility to retrieve a very large document with all the metadata and pages, by a single HBase row. This means that from an HBase table, the maximum amount of information can be retrieved by a request, which reduces server round trips. So, de-normalization improves the performance of the system, even if the server has to deal with many simultaneous requests. [14]

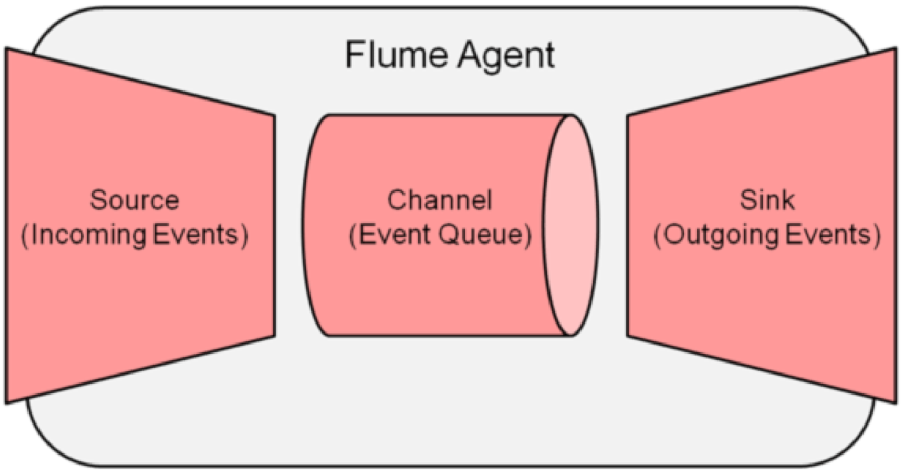
HBase partitions horizontally the tables into, the so called, **regions**. Every region has a set of table rows. As a start, a table is a single region, but when it grows and overcomes a specific threshold, the region is split into two new regions of almost equal size. Every region is hosted by a **Region Server**. Region servers run on the data nodes of a Hadoop cluster. As a HBase table grows, the number of the regions grows as well. This results in the distribution of the regions in the HBase cluster. So, if a table is very large to accommodate in one server, it breaks in regions which are distributed through a Hadoop cluster and every node – server hosts a subset of the whole table. [14]

Apart from the Region Server, there are some other vital components that HBase uses. **Apache Zookeeper** is one of them. It is the Hadoop coordinator for the distributed applications. **HBase Master** is another component which is responsible for the administrative operations, such as keeping track of the data of each node, according to the row key, control the load balancing through the cluster and managing any failover. There are many HBase Masters, but only one is actually the Master node and the others exist for backup, in case the Master fails. The choice of which node will be the Master is taken by the Zookeeper. [1], [14]

3.7 Apache Flume

The large dataset that Hadoop is processing, at first they were in another filesystem and then moved to HDFS. In many cases the data needs to be streamed in the HDFS, for reasons such as the analysis of data deriving from systems that produce only streams. When data is contained in high throughput streams, Apache Flume enables the aggregation, storage and analysis of this data with Hadoop. Flume is specialized for the insertion, in the Hadoop, of data produced during an event. Events such as application logs, social media updates or data retrieval from IoT sensor devices, are aggregated into new files in HDFS, in order to get processed. The destination of the files is called *sink,* in Flume parlance, and it is usually the HDFS. Apart from HDFS, Flume is flexible and can also write to other systems, such as HBase. [1]

Flume is available when it runs with a Flume *agent*, which is a Java process and chains all the components of the Flume architecture: *sources* connected via *channels* to *sinks*. Agents collect data from different applications and integrate this data into Hadoop.



(Copyright 2013, Dan McClary, drdobbs.com)

Figure 3.7.1: Flume agent's components [16]

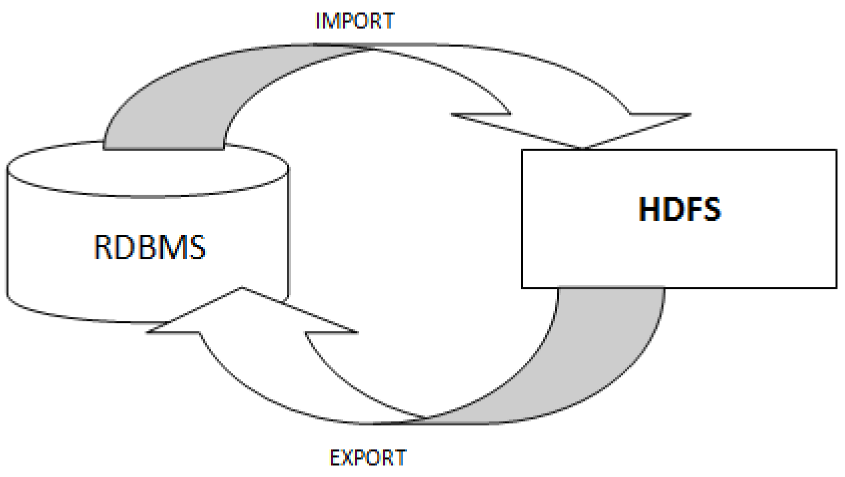
Sources in Flume are event producers. A Flume agent can have one or more sources. Each source has a name and a type. After the definition of these two, additional configuration can be achieved through the type. The events are delivered from the various sources to the channel. [1], [16]

Channel is a mechanism, which is responsible for storing the events and transferring them from their sources to the sinks. Events are removed from channels only when sinks command this action, which occurs when they have written successfully the data to an external repository, such as HDFS. Channels are distinguished in two types. The first category stores events in memory, offering high throughput, and the second concerns channels that are file or database-backed. In case of an agent failure, the first category cannot recover the data, but the second can not only recover but also reproduce the event. [1], [16]

Sinks plug to the output storage system, by writing a suitable Java class with the necessary classes. Such storage systems could be for instance HDFS and HBase. Sinks are responsible for the transaction of events to the output and after the success, to remove the written events from the channel. [1], [16]

3.8 Apache Sqoop

SQOOP is a combination of the words **SQ**L and Had**oop**. It is a tool that offers connectivity between relational databases or data warehouses with Hadoop. Users can specify the location in HDFS where the data being moved from RDBMS, such as MySQL, will be saved. Sqoop is also used for the extraction of data, out of Hadoop, into external structured datastores. [7], [8]



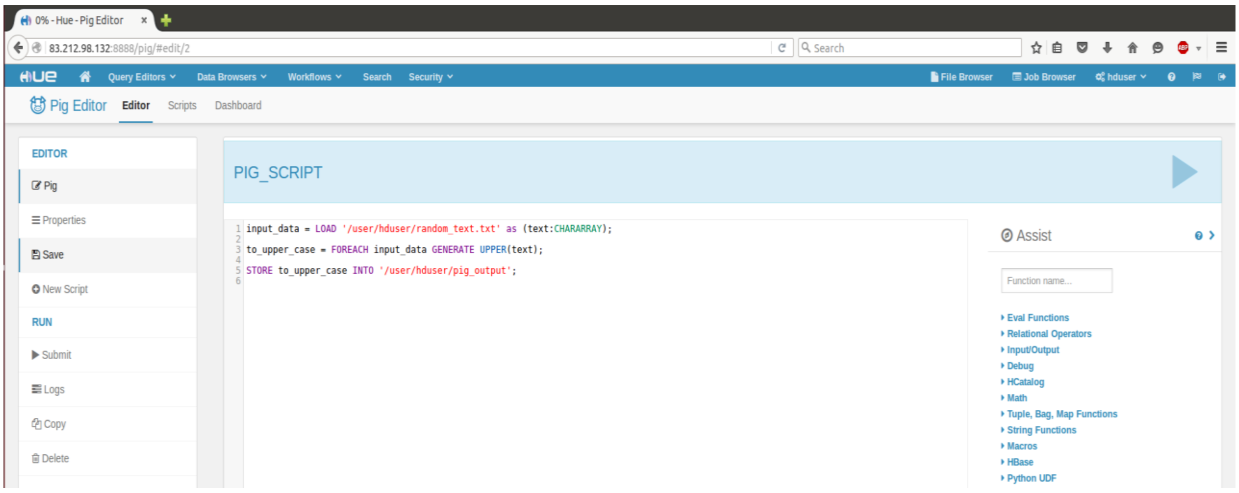
(Copyright Deepesh Kumbhare, HadoopTutorials.co.in)

Figure 3.8.1: Data transaction between RDBMS and HDFS [7]

Sqoop is used in order to import structured data from RDBMSs to HDFS. When this happens a file is created in HDFS representing the data to be processed, with a MapReduce or other program or component from the Hadoop ecosystem. After the data processing and analysis, the structured data is ready to be transferred and saved back to a table in a RDBMS through Sqoop. [7], [8]

3.9 Hue (Hadoop Web Interface)

Hue is an open-source Hadoop Web Interface, which provides a very friendly Graphical User Interface (GUI) for handling the transaction of data/files between the local disk and HDFS. Additionally, it offers a GUI for each component of the ecosystem. For example, for oozie, as mentioned to the corresponding section above, there is a drag and drop option to create workflows. It also includes editors and dashboards for other components, such as Pig, where the user can write scripts or hive, where the editors can be filled with SQL queries and execute them by pressing the related button.



```
input_data = LOAD '/user/hduser/random_text.txt' as (text:CHARARRAY);

to_upper_case = FOREACH input_data GENERATE UPPER(text);

STORE to_upper_case INTO '/user/hduser/pig_output';
```

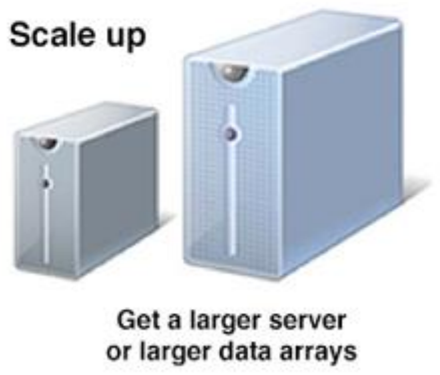
Figure 3.9.1: Pig script written in Hue's Pig Editor.

The above figure depicts the Hue's Pig Editor. For the sake of the example, I have written a Pig script, which takes as input (Loads) a text file, it places the words in an array and transforms the case of every word to uppercase. Finally, the result is saved in HDFS, to the specified directory. In order to execute the script, we press the play button in the right top corner of the screen. During the execution, a progress bar is displayed, showing the status of the job. If we open the oozie dashboard, we will see that the execution of the pig script is a single job, which is considered as a simple workflow.

4. Scale Hadoop Cluster

As data grows in a Hadoop Cluster, the performance of the processing and analysis, and the capacity is decreased. However, Hadoop can handle larger data by *scaling-up* or *scaling-out*. When there is need for extra storage or processing power, the two solutions mentioned will resolve any difficulties, even without changing the hardware of the existing machines at all, by taking advantage of the distributed nature of HDFS. This will be described below in the scaling-out paragraph. The non-distributed way of resolving the issue examined in this section, is to enhance the existing hardware.

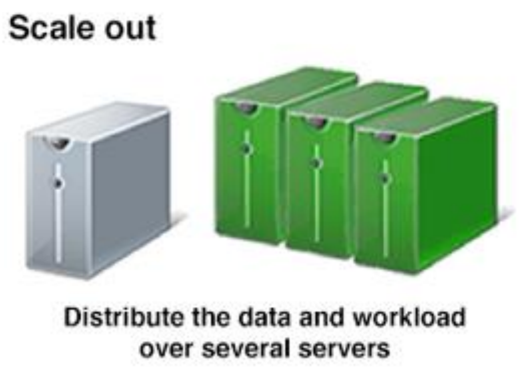
Scaling-up or **Vertical Scaling,** means to change the hardware, by increasing the CPU, RAM and/or Disk. This solution is not the most suitable in Big Data platforms because there are some limitations when changing the hardware. Even if the capacity of the physical machine increase enough to suit the larger hardware with the higher capabilities, there is a maximum technology limit, in terms of production. For instance, the disk capacity cannot increase above a limit in a physical machine. Furthermore, vertical scaling includes the purchasing and installing new hardware, which is not cost efficient. [17]



(Copyright 2016, Pivotal Software, Inc, Pivotal Documentation)

Figure 4.1: Vertical scaling of Hadoop cluster [18]

In contrast, **Scaling-out** or **Horizontal Scaling** is the addition of new nodes in the Hadoop Cluster, when it is necessary. This action does not require any change, software or hardware, to the platform, because it is actually a connection of the existing cluster of machines, with new ones. These new nodes, is not necessary to be high performance machines, in terms of characteristics such as CPU, as they will be integrated with the rest Hadoop Cluster and the total performance will be increased. Every new node is another commodity hardware added in the network ready to offer its computational abilities for data processing. Horizontal scaling takes advantage of the distributed processing, which is the reason why it harmonizes more with the Hadoop concept. [17]



(Copyright 2016, Pivotal Software, Inc, Pivotal Documentation)

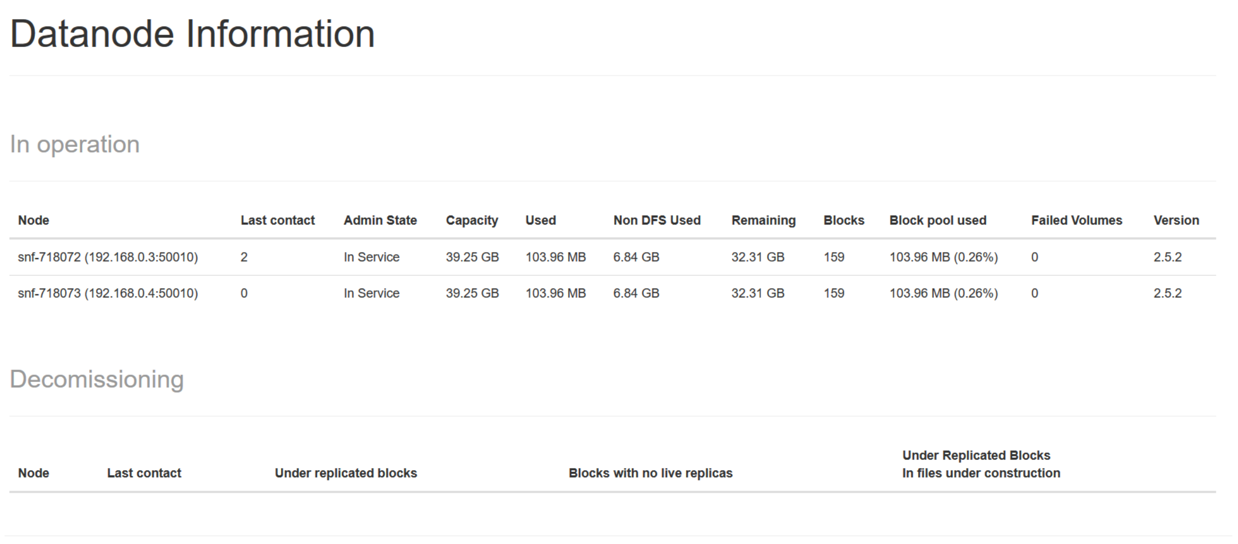
Figure 4.2: Horizontal scaling of Hadoop cluster [18]

Hadoop framework is empowered by the distributed characteristic, so we will focus on the horizontal scaling, which follows the distributed nature of HDFS. In the next sections it will be analysed the exact way of adding and removing nodes to/from an existing Hadoop Cluster.

4.1 Adding nodes to an existing Hadoop Cluster

The procedure of adding a node, and more precisely a Name Node, to an existing Hadoop Cluster, in order to increase the storage and the computing capabilities, has been investigated and will be analysed in this section. The analysis is based on specific steps, which are going to be applied to a three node Hadoop Cluster, created for the specific master thesis for the proof of concept. The procedure of creating a Hadoop Cluster is not in the scope of this section, so it is not included, but it is going to be analysed in the next chapter.

Concerning the node addition, the main purpose described below is to add a Name Node to the three node Hadoop Cluster. Initially, by hitting the URL http://<IP>:50070 , where the <IP> is the IP (Internet Protocol) of the Hadoop cluster's Name Node, an overview about the cluster is displayed in the screen, and by choosing the "Datanodes" tab the following result is produced.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
snf-718072 (192.168.0.3:50010)	2	In Service	39.25 GB	103.96 MB	6.84 GB	32.31 GB	159	103.96 MB (0.26%)	0	2.5.2
snf-718073 (192.168.0.4:50010)	0	In Service	39.25 GB	103.96 MB	6.84 GB	32.31 GB	159	103.96 MB (0.26%)	0	2.5.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Figure 4.1.1: Initial Name Node Information and status.

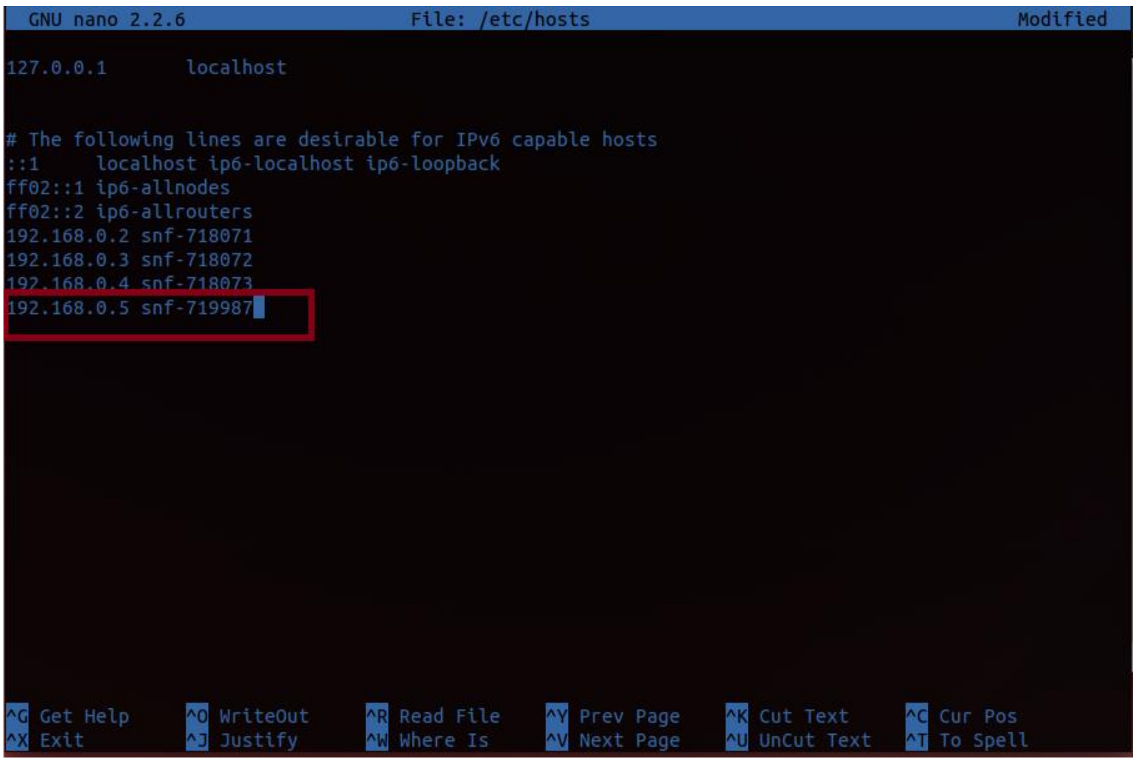
The picture depicts the information about the Name Nodes, accessed by the Name Node of the cluster. There are two sections: "In operation" and "Decommissioning". The first section is about the active Name Nodes with information about their hostname, capacity, versions etc. The hostname is "snf-<id>" ("snf" stands for "synnefo"), because the Hadoop Cluster is composed of virtual machines (VMs) created on GRNET's ~Okeanos IaaS (Infrastructure-as-a-Service), powered by "Synnefo" (Greek word for "Cloud") which is an open source cloud software designed and developed by GRNET. The "Version" is referred to the Hadoop version, which is the 2.5.2.

1) The first step of the procedure is to create a new Virtual Machine from ~Okeanos IaaS and install a Hadoop distribution, same as the rest nodes of the existing cluster, which in our case is a Hadoop 2.5.2. The created machine is allowed to have greater, lower or the same storage, and more, fewer or the same number of CPUs. The only

feature that is of great concern is the memory, which is important to not be lower than the preconfigured memory in the Hadoop XML files.

2) During the first step it is important to add the created machine to the cluster's network. This can be achieved in our case through the ~okeanos GUI (Graphical User Interface). The new machine will have an internal IP, in our case 192.168.0.5, as the rest nodes start from 192.168.0.2 to 192.168.0.4. The external IP, from which the cluster can be accessed, resides in the Name Node.

3) The next step is to update the Name Node's "hosts" file, which is in the location "/etc/hosts", with the new Name Node's private IP and hostname.



```
  GNU nano 2.2.6                  File: /etc/hosts                          Modified

127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.0.2 snf-718071
192.168.0.3 snf-718072
192.168.0.4 snf-718073
192.168.0.5 snf-719987

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Figure 4.1.2: Adding new Name Node's IP and hostname in "hosts" file of every node.

As the figure depicts, the /etc/hosts file has opened with the "nano" editor and the line in the red box has been added. The private IP of the new Name Node is 192.168.0.5 and the hostname is snf-719987. The "hosts" file should be the same in every node.

4) Continuing, it is necessary to perform a rerouting from the Name Node to the Name Node, which is achieved with port forwarding in the Name Node machine. The commands that fulfil the above state are "*iptables -A PREROUTING -t nat -i eth1 -p tcp --dport 10002 -j DNAT --to 192.168.0.5:22*" and "*iptables -A FORWARD -p tcp -d 192.168.0.4 --dport 22 -j ACCEPT*" which are executed in the command line of the Name Node. The ports 10000 and 10001 are already bound by the first two Name Nodes, so in the above command we bind the port 10002 for the new Name Node. The Name Node will try to find the new Name Node to the given internal IP at port 22 (which is the port forwarding action). More specifically, every request in Name Node's port 10002 will be rerouted to the new Name Node at port 22, which is achieved through a translation offered by the DNAT (Destination Network Address Translation) method. Also, we must perform an action to the Name Node, so as to receive the incoming requests at port 22. By running the command "*route add default gw 192.168.0.2*", we enable a link between the Name Node, which has the stated internal IP, and the Name Node. This link between those two machines is also known as gateway.

5) In the new Name Node we create a user "hduser" and a "hadoop" group. Also, for our convenience we rename the directory "/usr/local/hadoop-2.5.2" to "/usr/local/hadoop". Finally, we change the owner of this directory to hduser, by running the command "*chown –R hduser:hadoop /usr/local/hadoop*".

6) The next step is divided in two parts. The first one is to navigate to the "/usr/local/hadoop/etc/hadoop" directory and modify the "slaves" file, by adding the hostname of the new added Name Node into it. This action must be performed in the Name Node of the cluster. It should be noted, as a reminder, that "slaves" is another word for referring to the Name Nodes. The second part of this step is to copy all the configuration XML files from the "/usr/local/hadoop/etc/hadoop" directory of Name Node to the same directory of the newly added Name Node. The "slaves" file in the new added data node should only contain the word "localhost".

7) The Name Node needs to interact with the Name Nodes many times during a

workflow. Every machine is protected by a password. In order to avoid typing the password of every node for every interaction, we use ssh (Secure SHell) keys, which are the intermediate to operate network services securely over the Hadoop cluster's network, with the use of ssh cryptographic network protocol. These keys are saved in a file in every machine. In the "~/.ssh" directory the file "authorized_keys" has the ssh keys. We copy this file from the Name Node to the Name Node, so as to achieve a trusted communication between these two machines, without password requirements. This action should be performed by the hduser.

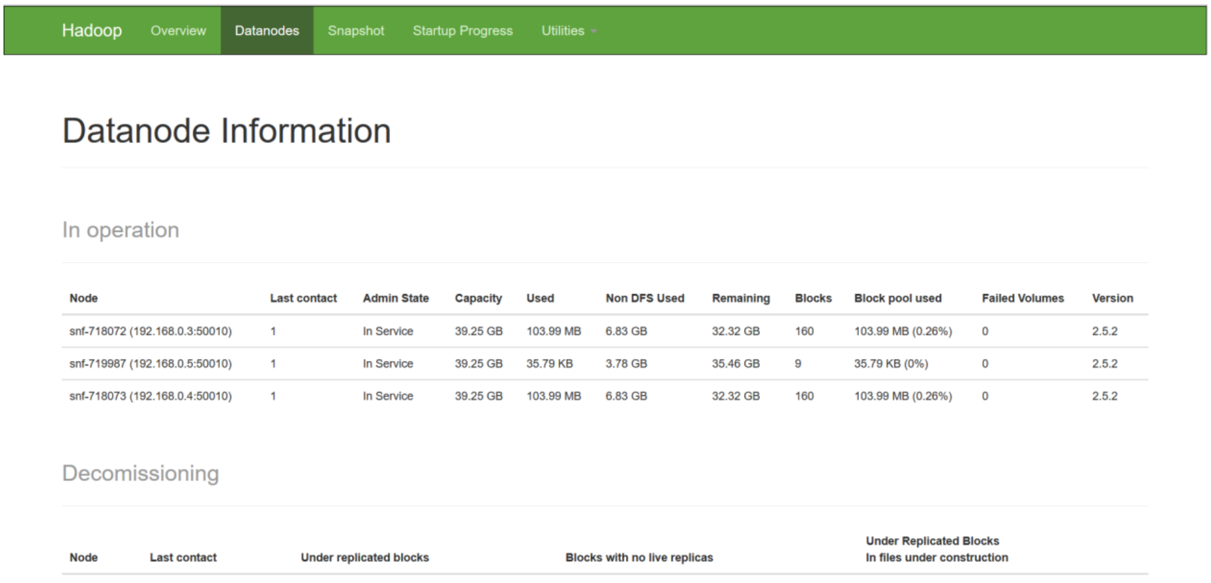
8) In the newly added Name Node, it is important for the directories "/app/hadoop/tmp", "/app/hadoop/tmp/namenode" and "/app/hadoop/tmp/datanode" (if they do not exist they should be created) to have appropriate permissions and owner. In order to change the owner to "hduser" and the group to "hadoop", we run the command "*chown –R hduser:hadoop /app/hadoop/tmp*" and the same for the other two directories. Furthermore, to change the permissions to these directories we run the command "*chmod 750 /app/hadoop/tmp*" and the same for the other two directories. The number 750 indicates that the hduser can fully modify these directories, the hadoop group can read and access these directories, and the rest users have no access to these directories.

9) Continuing, the file "regionservers" in the ""/usr/local/hadoop/etc/hadoop" directory should be updated with the hostname of the new added node. This action must be performed in every node of the cluster. The last file to be configured is the "include" file, in the same directory as regionservers. This file will be updated only in the Name Node of the cluster, with the hostname of the new data node.

10) Finally, a Hadoop Cluster restart is required in order to apply the configurations across the Hadoop Cluster. In the Name Node we run the following commands as hduser. Firstly, we stop the distributed filesystem by typing "stop-dfs.sh". Then, we stop the YARN: "stop-yarn.sh", and the job history server: "mr-jobhistory-daemon.sh stop historyserver". These commands are actually UNIX shell scripts that run in the

background many tasks. In order to start again the Hadoop services we rerun the above commands by replacing the word stop with start.

After following the above instructions a new data node has successfully been added to the initial three node cluster, created for this master thesis. The nodes have increased to four, from which the three nodes are slaves, or data nodes, and the other one is the master, or the name node.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
snf-718072 (192.168.0.3:50010)	1	In Service	39.25 GB	103.99 MB	6.83 GB	32.32 GB	160	103.99 MB (0.26%)	0	2.5.2
snf-719987 (192.168.0.5:50010)	1	In Service	39.25 GB	35.79 KB	3.78 GB	35.46 GB	9	35.79 KB (0%)	0	2.5.2
snf-718073 (192.168.0.4:50010)	1	In Service	39.25 GB	103.99 MB	6.83 GB	32.32 GB	160	103.99 MB (0.26%)	0	2.5.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Figure 4.1.3: Name Node Information and status after the addition of the new node.

Comparing the above figure with the 4.1.1, we can see that the newly added data node is recognised by the Name Node and it is an active part of the Hadoop Cluster. As a result, the processing power of the Hadoop has increased, along with the storage of the HDFS. [19]

4.2 Removing nodes from a Hadoop Cluster

One of the greatest advantages of the Hadoop framework is that it utilizes commodity hardware, reducing the costs of enterprise companies, which analyse huge amounts of data with many machines working in parallel. However, this does not offer stability in a Hadoop

Cluster, as the data nodes that run many tasks may crash irrecoverably. In that case the Hadoop framework gives a handy solution. The removal of the crashed data node is a procedure where the problematic machine is decommissioned from the cluster, without losing any blocks of data stored in HDFS. Before the procedure of decommission, the data blocks are rebalanced to the existing nodes so as not to get lost. Below it is described analytically the procedure of removing (decommissioning) a Name Node from the Hadoop Cluster. For the proof of instructions, the four node (after the node addition) cluster will be used for the execution of the commands and the appliance of the configuration files.

1) The first step of the procedure is to add a list of the hostnames of the data nodes, which are necessary to be decommissioned, to the “exclude” file located in the “/usr/local/hadoop/etc/hadoop” directory. This action will take place in the Name Node exclusively. In our case we assume that the data node that we no longer need is the node with the hostname “snf-718073” and internal network IP “192.168.0.4”.

2) The next step is to update two configuration files of the Name node, located in the “/usr/local/hadoop/etc/hadoop” directory. The first one is the “hdfs-site.xml”, which should be updated with the property “dfs.hosts.exclude” that informs the name node about the nodes to be decommissioned. The lines to be added are:

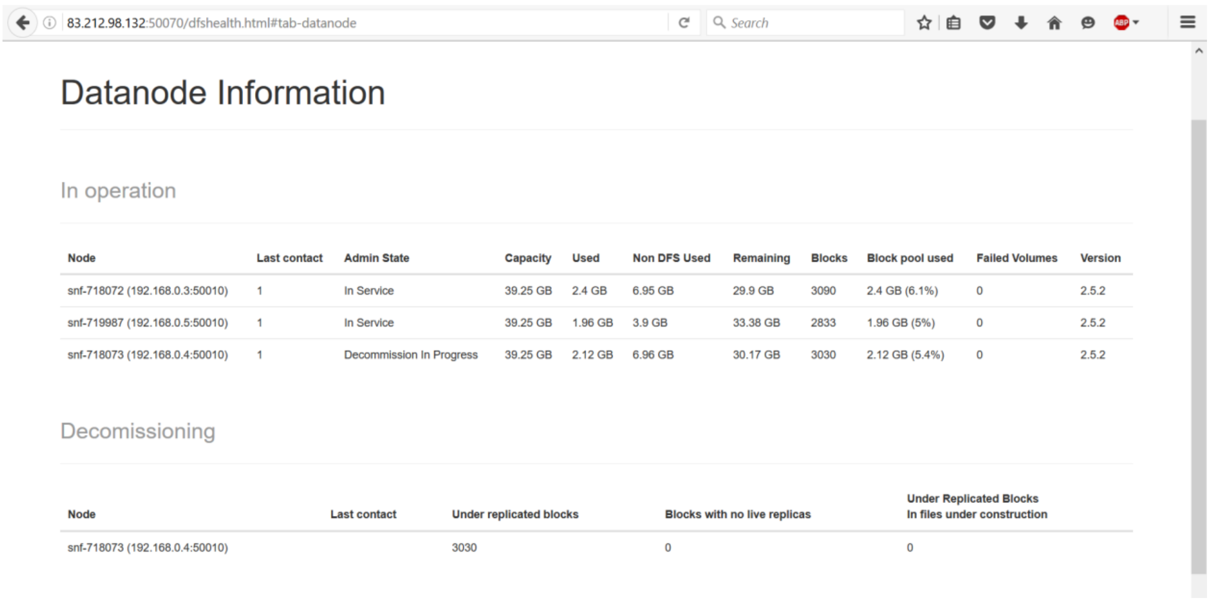
```
<property>
      <name>dfs.hosts.exclude</name>
      <value>/usr/local/hadoop/etc/hadoop/exclude</value>
      <description> List of nodes to decommission </description>
</property>
```

   In this way the HDFS knows which hostnames will be excluded from the Hadoop cluster. The second configuration file to be updated is the “mapred-site.xml” and the property that should be added is the “mapred.hosts.exclude”. The lines to be added are:

```
<property>
      <name>mapred.hosts.exclude</name>
      <value>/usr/local/hadoop/etc/hadoop/exclude</value>
      <description> List of nodes to decommission </description>
</property>
```

3) The name node will get updated by running the command “*hadoop dfsadmin –refreshNodes*”. After executing this command the name node will start the

decommission process for the data nodes included in the “exclude” file.



83.212.98.132:50070/dfshealth.html#tab-datanode

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
snf-718072 (192.168.0.3:50010)	1	In Service	39.25 GB	2.4 GB	6.95 GB	29.9 GB	3090	2.4 GB (6.1%)	0	2.5.2
snf-719987 (192.168.0.5:50010)	1	In Service	39.25 GB	1.96 GB	3.9 GB	33.38 GB	2833	1.96 GB (5%)	0	2.5.2
snf-718073 (192.168.0.4:50010)	1	Decommission In Progress	39.25 GB	2.12 GB	6.96 GB	30.17 GB	3030	2.12 GB (5.4%)	0	2.5.2

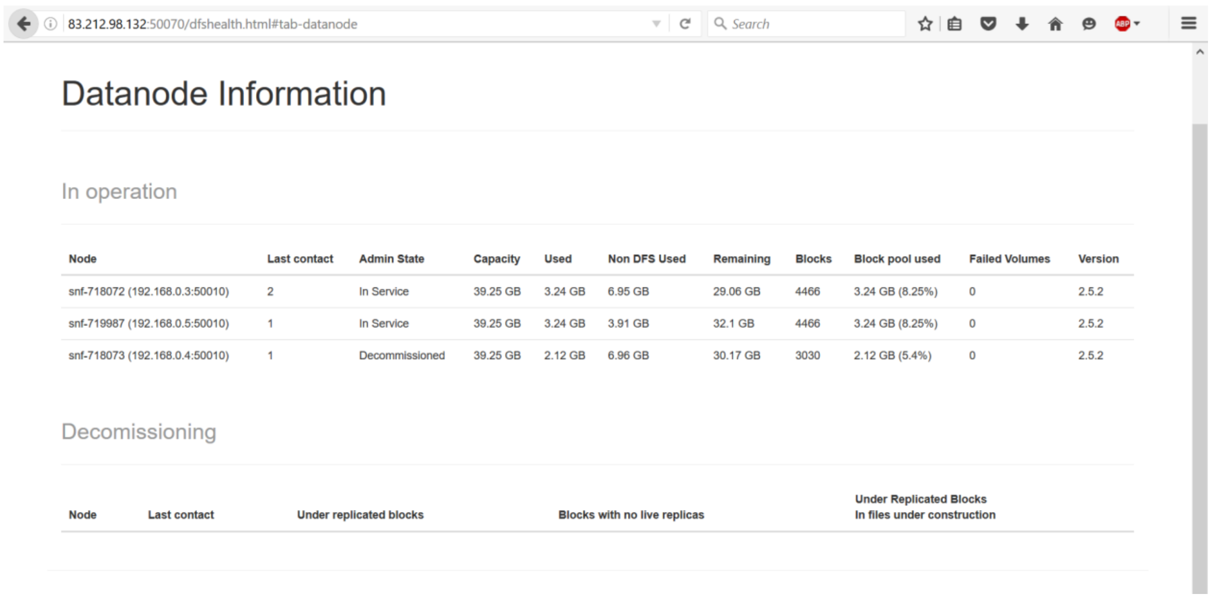
Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
snf-718073 (192.168.0.4:50010)		3030	0	0

Figure 4.2.1: Decommissioning of Data Node in Progress.

The above figure depicts the administration UI of the Name Node, at the PORT 50070. We can see that the state of the data nodes included in the “exclude” list has changed to “Decommission in Progress”, which in our case is the “snf-718073”. The background task running in a decommissioning node comprises of the copy of its blocks to other data nodes. This process may take from minutes to hours to complete, depending on the data volumes of the decommissioned node. During the decommissioning procedure, the column “Under replicated blocks” number decreases, according to the replication of the blocks to the other data nodes.

4) When a data node completes the replication of the blocks to the other data nodes of the cluster, it reports its state as “Decommissioned” which indicates that the node is ready to leave the cluster with no danger for data loss.


83.212.98.132:50070/dfshealth.html#tab-datanode

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
snf-718072 (192.168.0.3:50010)	2	In Service	39.25 GB	3.24 GB	6.95 GB	29.06 GB	4466	3.24 GB (8.25%)	0	2.5.2
snf-719987 (192.168.0.5:50010)	1	In Service	39.25 GB	3.24 GB	3.91 GB	32.1 GB	4466	3.24 GB (8.25%)	0	2.5.2
snf-718073 (192.168.0.4:50010)	1	Decommissioned	39.25 GB	2.12 GB	6.96 GB	30.17 GB	3030	2.12 GB (5.4%)	0	2.5.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

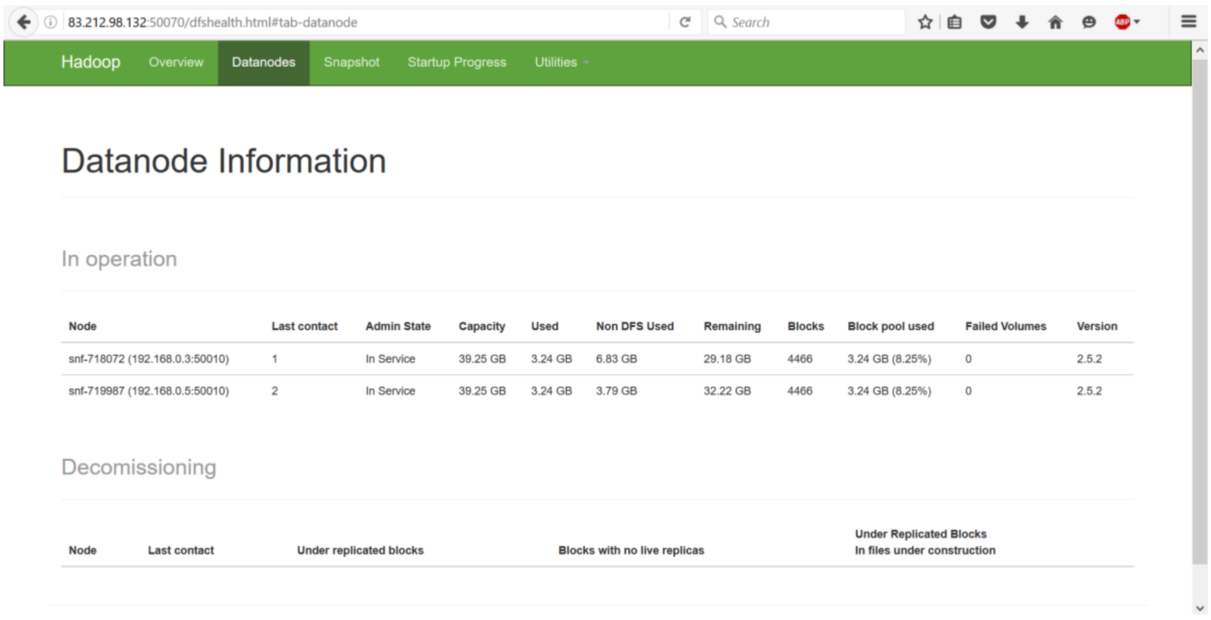
Figure 4.2.2: Data Node decommissioned.

Afterwards, the decommissioned node is no longer part of the Hadoop cluster and the user can shut down or restart the machine.

5) The next step is to remove the hostname of the decommissioned data node from the “slaves” file, which is in the directory “/usr/local/hadoop/etc/hadoop”. Also, in the same folder the file “include” needs to get updated by removing the hostname, same as the “slaves” file. These two actions should happen only in the Name Node. The last file that should be updated is the “regionservers” in the same folder as the two previous files mentioned, in which we perform the same action, but for every node of the Hadoop Cluster.

6) In order to refresh the nodes, it is necessary to run the following command in the name node as user “hduser”: *hadoop dfsadmin –refreshNodes*. After the refresh the command “*start-balancer.sh*” will perform a sharing and reallocation of the blocks data, to the available Data Nodes. Last but not least command, is the “*hadoop fsck –blocks*” which will show the status of the data replication.

7) Finally, a Hadoop Cluster restart would be required, so as to make certain that the Hadoop Cluster is fully aware of the removal of the Data Node.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
snf-718072 (192.168.0.3:50010)	1	In Service	39.25 GB	3.24 GB	6.83 GB	29.18 GB	4466	3.24 GB (8.25%)	0	2.5.2
snf-719987 (192.168.0.5:50010)	2	In Service	39.25 GB	3.24 GB	3.79 GB	32.22 GB	4466	3.24 GB (8.25%)	0	2.5.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Figure 4.2.3: Data Node decommissioned and removed successfully.

In the above figure it is clearly depicted that by following the steps of this subsection, a Hadoop administrator could remove successfully a Data Node from a Hadoop Cluster. In this way, any crashed machine in the cluster, could get removed and replaced by a new healthy machine, which can be added easy according to the steps described in the previous subsection. [19]

5. Installation of testing environment

In this chapter it is presented a Hadoop installation in three virtual machines from which, the one will be the Name Node of the Hadoop Cluster and the other two are going to be the Data Nodes. Afterwards, some components of the Hadoop Ecosystem are going to be added, such as oozie (see chapter 3.3) and pig (see chapter 3.2), so as to create a simple workflow that will run a map/reduce job and some pig scripts. The purpose is to count the execution time of a workflow, before and after the addition of a Data Node, so as to examine the processing power and response time offered by an extra node in a Hadoop Cluster.

5.1 Hadoop installation

Initially, it will be presented the installation of the Hadoop framework as a step by step procedure. As a start, we create as many virtual machines as the desired size of the cluster. The operating system chosen for the Hadoop installation in this master thesis is Debian Linux. All the machines are in the same internal network and only the machine that will be the Name Node of the cluster has an external IP. In every machine we repeat the following procedure.

1) The installation of Oracle Java 8 programming language is required, as the Hadoop is a Java framework.

2) Linux systems support groups and users, so we created a “Hadoop” group and an “hduser” user to administrate and control the Hadoop related operations.

3) The next step was to install the openssh-server, in order to allow secure communication between the nodes of the Hadoop Cluster, without the prompt for password, as the interactions will be many during the workflows. After the installation, it is important to generate a ssh key for the hduser and copy the id_rsa.pub to the “authorized_keys” file, which is in the “~/.ssh” directory.

4) Continuing, we disable the IPv6 by configuring (or adding) in the “sysctl.conf” file, which is in the “/etc” directory, the lines:

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

5) Afterwards, we downloaded (for the installation of this master thesis) the Hadoop-2.5.2.tar.gz and we extracted it to the folder “/usr/local”. We rename the extracted folder from hadoop-2.5.2 to hadoop, for convenience, and we create some temporary directories for the name node and the data nodes.

6) Concerning the configurations, we need to update the file “~/.bashrc” with some environmental Hadoop variables. So, we add the next lines to the file:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```

Also, to the file “hadoop-env.sh” we need to write the java home path, so we add the line:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

We create the files “slaves”, “includes” and “regionservers” in the Name Node, in the “/usr/ local/hadoop/etc/hadoop/” directory, and we write inside them the hostnames of the Data Nodes of the cluster.

In the “/usr/local/hadoop/etc/hadoop/” directory we open the “core-site.xml” configuration file and we add the lines:

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <final>true</final>
  <description>A base for other temporary directories.</description>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://snf-718071:9000</value>
  <final>true</final>
</property>
```

The “snf-718071” shown above, is the hostname of the Name Node.

In addition, in the same directory we add to the “hdfs.site.xml” file the lines:

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/app/hadoop/tmp/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/app/hadoop/tmp/datanode</value>
</property>
<property>
  <name>dfs.blocksize</name>
  <value>128m</value>
  <description>Block size</description>
</property>
<property>
  <name>dfs.hosts</name>
  <value>/usr/local/hadoop/etc/hadoop/include</value>
  <description> List of nodes that connect to namenode </description>
</property>
```

Above we can see the paths to the temporary directories for name node and data nodes, the size of a block of data that is going to be saved in the HDFS nodes and the path to the file (includes) which has a list with the nodes that connect with the name node.

In the “yarn-site.xml” file we configure the resource manager by adding:

```
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>snf-718071</value>
  <final>true</final>
  <description>host is the hostname of the resource manager.
  </description>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>snf-718071:8025</value>
  <final>true</final>
  <description>host is the hostname of the resource manager and
   port is the port on which the NodeManagers contact the Resource
Manager.
  </description>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>snf-718071:8030</value>
  <final>true</final>
  <description>host is the hostname of the resourcemanager and port
is the port on which the Applications in the cluster talk to the
Resource Manager.
```

```
  </description>
</property>
```

In the configurations above we can see that the resource manager's hostname and ports are defined. The resource manager derives in the Name Node and the ports 8025 and 8030 are responsible for the tracking and the scheduling of the resources for the execution of a workflow accordingly.

And finally, we verify that in the "mapred-site.xml" file the next lines are present:

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

7) When the configuration is completed, we format the name node and we start all the Hadoop daemons by running the following commands in the name node's terminal, as user hduser:

```
hdfs namenode –format
start-dfs.sh
start-yarn.sh
mr-jobhistory-daemon.sh start historyserver
```

8) By running the command "jps" in the Name Node's terminal, we can see a list of the running services and verify if everything is running as expected.

The above steps performed in three machines and the final result was a fully functional three node Hadoop Cluster. In order to create a folder for the hduser in the HDFS we run the commands "hdfs dfs –mkdir /user" and "hdfs dfs –mkdir /user/hduser".

5.2 Installing Hue on Hadoop

Hue is a Hadoop Web Interface and its friendly Graphical User Interface (GUI) will help us to create oozie workflows and execute them. The installation is comprised by the steps described below.

1) Initially, we connect to the Name Node and we install the required dependencies by

running the following commands:

```
sudo apt-get update
sudo apt-get install -y ant
sudo apt-get install -y gcc g++
sudo apt-get install -y libkrb5-dev libmysqlclient-dev
sudo apt-get install -y libssl-dev libsasl2-dev libsasl2-modules-
gssapi-mit
sudo apt-get install -y libsqlite3-dev
sudo apt-get install -y libtidy-0.99-0 libxml2-dev libxslt-dev
sudo apt-get install -y maven
sudo apt-get install -y libldap2-dev
sudo apt-get install -y python-dev python-simplejson python-
setuptools
```

2) The next step is to download Hue with the command "wget https://dl.dropboxusercontent.com/u/730827/hue/releases/3.8.0/hue-3.8.0.tgz"

3) After the download is complete we unzip the file and in the extracted folder we run the command "sudo make install" in order to start the Hue installation.

4) Afterwards we create a user and a group "hue". Also, we change the ownership of the "/usr/local/hue" directory to the newly created user and group.

5) The Hadoop configurations are next, which will happen to the "hdfs-site.xml" and "core-site.xml" files. In the first file we add the lines:

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>
```

   And in the second file we add the lines:

```
<property>
  <name>hadoop.proxyuser.hue.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.hue.groups</name>
  <value>*</value>
</property>
```

   Then, we copy these two files to all data nodes of the Hadoop Cluster.

6) In addition, we configure the hue.ini file, which is in the location

"/usr/local/hue/desktop/conf". Firstly, we uncomment the line "default_hdfs_superuser=hduser" so as to add the hduser as superuser. Secondly, we set the Name Node's IP to the "fs_defaultfs" and "webhdfs_url" sections. Also, we set the IP of the resource manager's host, which in our case is the Name Node, to the sections "resourcemanager_host" and "resourcemanager_api_url" and the same for the sections "proxy_api_url" and "history_server_api_url".

7) Finally, we start the Hue service on Hadoop by running the commands:

```
su - hue
cd /usr/local/hue/build/env/bin/
./supervisor -d
```

As hue user we start the Hue daemon.

After following the above steps, the Hue will get installed to the Hadoop and it will be available at the Name Node's IP in the port 8888. It is important to login in Hue as user hduser so as to have full access to the HDFS. [20]

5.3 Installing pig and oozie components

In this subchapter it will be described the procedure of how to install oozie in a Hadoop Cluster. For more information about oozie see chapter 3.3. The oozie component will be used for the creation of workflows in the next chapter. It is necessary to install Pig before oozie so we will start describing the installation steps for this component first.

5.3.1 Pig installation

For the Pig installation, it is only needed to download the compressed file, that includes the necessary libraries and execution files, and add it to the Linux PATH. The Linux PATH is an environmental variable, which reports to UNIX operating systems "shell" in which directories it should search to find the executable files when a user types a specific command. The steps for the installation are to change user to hduser by typing in terminal “su hduser” and download Pig with the command “wget http://mirrors.myaegean.gr/apache/pig/pig-0.14.0/pig-0.14.0.tar.gz”. The next step is to unzip the downloaded file by typing “tar -zxvf pig-0.14.0.tar.gz”. Finally, we add Pig program in the PATH with the command “export PATH=/home/hduser/pig-0.14.0/bin/:$PATH”, so as to open the Pig editor in the Linux terminal by typing only the command “pig”.

5.3.2 Oozie installation

The oozie installation is more complex than Pig’s, because there are some configuration requirements that need to be fulfilled. In the steps below, it is described the procedure analytically.

1) Initially, we download oozie from a mirror with the command “wget http://mirrors.myaegean.gr/apache/oozie/4.1.0/oozie-4.1.0.tar.gz”. We unzip the downloaded file by running “tar -xvzf oozie-4.1.0.tar.gz” and we access the oozie folder: “cd oozie-4.1.0”.

2) The second step is to build oozie, which will be achieved by running the command “mvn clean package assembly:single -P hadoop-2 -DskipTests”. The “mvn clean” command tells Maven to build all the modules and install them in the local directory of the machine running the command. Maven is a project management tool that helps and makes more convenient the installation of projects/programs that have many dependencies. The option “-P hadoop-2” means that we choose the profile as hadoop 2, because we currently have installed Hadoop 2 in our virtual machines.

3) Thereafter, it is necessary to setup the oozie server. We create a folder “Oozie” and we recursively copy the “oozie-4.1.0” from the “distro/target/oozie-4.1.0-distro” directory into the created folder, with the command “cp -R distro/target/oozie-4.1.0-distro/oozie-4.1.0/ Oozie/”. Then, we change directory (cd) into “Oozie/oozie-4.1.0” and create a folder “libext”, where we will copy Hadoop libraries, by typing: “cp -R

../../hadooplibs/hadoop-2/target/hadooplibs/hadooplib-2.3.0.oozie-4.1.0/* libext/". We access the libext directory and we download one more library with the command: "wget http://dev.sencha.com/deploy/ext-2.2.zip".

4) For this step we need to install zip in the system, so we run the command "sudo apt-get install zip". After the installation, we type in terminal "bin/oozie-setup.sh prepare-war", which runs an oozie shell script that setups and prepares WAR (Web Archive) files for oozie.

5) Afterwards, we update the configuration files. As a start, we edit the "/usr/local/hadoop/etc/hadoop/core-site.xml" file and we add the lines:

```
<property>
  <name>hadoop.proxyuser.hduser.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.hduser.groups</name>
  <value>*</value>
</property>
```

   We save the changes and we copy the file to all data nodes of the cluster. Following, we create a directory "/etc/oozie/conf" and a file "oozie-site.xml" which should be completed with the lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
    <name>oozie.service.ProxyUserService.proxyuser.hue.hosts</name>
    <value>*</value>
</property>
<property>
    <name>oozie.service.ProxyUserService.proxyuser.hue.groups</name>
    <value>*</value>
</property>
</configuration>
```

   When the configurations are completed, we restart the HDFS.

6) The next step is to create a share library directory that oozie needs in order to operate. In order to achieve this, we change directory to the path "/home/hduser/oozie-4.1.0/Oozie/oozie-4.1.0/bin" and we run the command "sudo

-u hduser ./oozie-setup.sh sharelib create -fs hdfs://<IP>:9000”, where the <IP> is the address of the Name Node.

7) On the HDFS we create the directory oozie by running as hduser “hdfs dfs -mkdir /user/oozie” and we move the oozie share libraries to the default folder, which is the one just created, with the command “hdfs dfs -mv /user/hduser/share /user/oozie/”.

8) Another oozie requirement is to create an oozie database, which is achieved by simply finding the location of the “ooziedb.sh” shell script and running the command “./ooziedb.sh create -sqlfile oozie.sql -run”.

9) Finally, we start the oozie server by running the “oozied.sh” shell script, that deploys the oozie daemon, and giving as parameter the word “start”.

10) An optional step, but necessary for our case, is to update the “hue.ini” file with some extra configurations, in order to recognize that the oozie component is part of the current Hadoop ecosystem. More specifically, we uncomment and update the section of the “hue.ini” file as it is depicted below:

```
# Webserver runs as this user
server_user=hduser
server_group=hduser

# This should be the Hue admin and proxy user
default_user=hduser
```

    Also, in the section “[liboozie]” we uncomment the “oozie_url= http://<IP>:11000/oozie” and we replace the <IP> with the address of the Name Node.

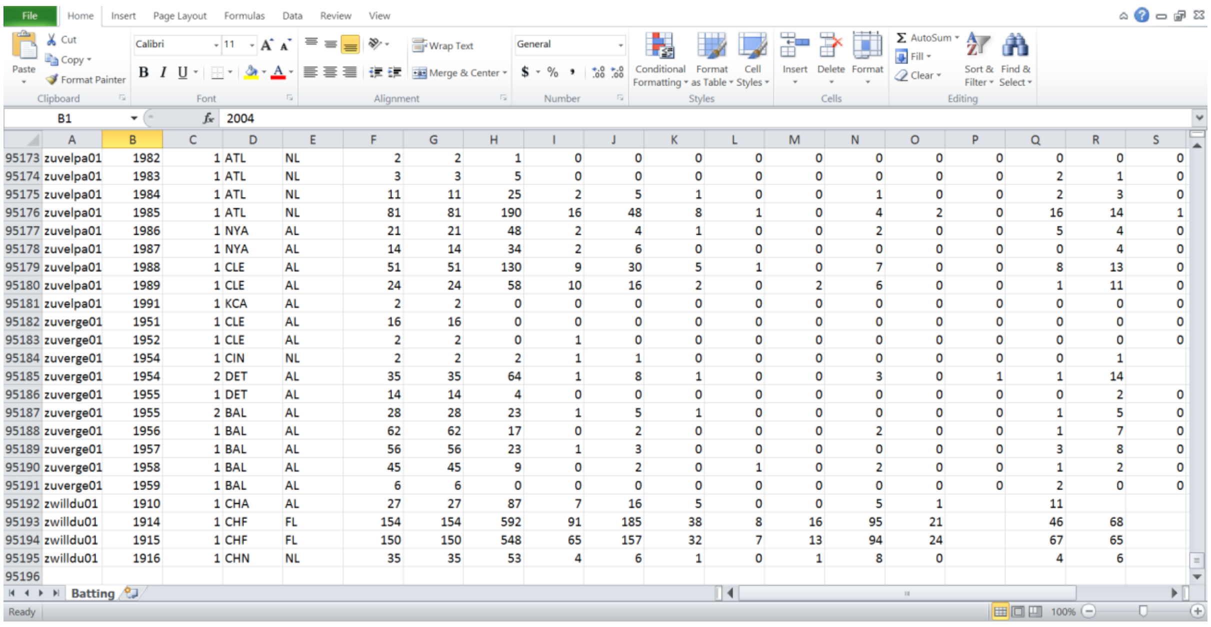
When the above installations are completed, we will have a Hadoop Cluster with a web graphical user interface and two components of the Hadoop ecosystem enabled. In the next chapter we will proceed to the creation of workflows, by using the Hue oozie editor.

6. Experiment – Workflows

This chapter will focus on the creation of a workflow with Apache oozie and the execution of a Map Reduce job through the Name Nodes console. The workflow will include some Pig scripts. We will use Hue's graphical user interface in order to create the workflow. The purpose is to create a workflow, run it in a four node Hadoop Cluster and examine the time required for the completion of the workflow. Afterwards, we will remove a node from the existing four node cluster and rerun the workflow, in order to examine the increment of time for the completion of the workflow and the task distribution over the decreased cluster. The same time examination will happen for the Map Reduce job.

6.1 Evaluation – Results on a four node cluster

Initially, we download a compressed file that includes many CSV (Comma-separated values) data files from the link http://hortonassets.s3.amazonaws.com/pig/lahman591-csv.zip, in order to choose one and process it with pig scripts. We choose the Batting.csv file, which has 95195 lines with many columns, making it difficult to be processed in a fast way. The file includes players and their successful shots over the years. Our purpose is to find the player with the most successful shots per year.



B1 2004

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
95173	zuvelpa01	1982	1	ATL	NL	2	2	1	0	0	0	0	0	0	0	0	0	0	0
95174	zuvelpa01	1983	1	ATL	NL	3	3	5	0	0	0	0	0	0	0	0	2	1	0
95175	zuvelpa01	1984	1	ATL	NL	11	11	25	2	5	1	0	0	1	0	0	2	3	0
95176	zuvelpa01	1985	1	ATL	NL	81	81	190	16	48	8	1	0	4	2	0	16	14	1
95177	zuvelpa01	1986	1	NYA	AL	21	21	48	2	4	1	0	0	2	0	0	5	4	0
95178	zuvelpa01	1987	1	NYA	AL	14	14	34	2	6	0	0	0	0	0	0	0	4	0
95179	zuvelpa01	1988	1	CLE	AL	51	51	130	9	30	5	1	0	7	0	0	8	13	0
95180	zuvelpa01	1989	1	CLE	AL	24	24	58	10	16	2	0	2	6	0	0	1	11	0
95181	zuvelpa01	1991	1	KCA	AL	2	2	0	0	0	0	0	0	0	0	0	0	0	0
95182	zuverge01	1951	1	CLE	AL	16	16	0	0	0	0	0	0	0	0	0	0	0	0
95183	zuverge01	1952	1	CLE	AL	2	2	0	1	0	0	0	0	0	0	0	0	0	0
95184	zuverge01	1954	1	CIN	NL	2	2	2	1	1	0	0	0	0	0	0	0	1	
95185	zuverge01	1954	2	DET	AL	35	35	64	1	8	1	0	0	3	0	1	1	14	
95186	zuverge01	1955	1	DET	AL	14	14	4	0	0	0	0	0	0	0	0	0	2	0
95187	zuverge01	1955	2	BAL	AL	28	28	23	1	5	1	0	0	0	0	0	1	5	0
95188	zuverge01	1956	1	BAL	AL	62	62	17	0	2	0	0	0	2	0	0	1	7	0
95189	zuverge01	1957	1	BAL	AL	56	56	23	1	3	0	0	0	0	0	0	3	8	0
95190	zuverge01	1958	1	BAL	AL	45	45	9	0	2	0	1	0	2	0	0	1	2	0
95191	zuverge01	1959	1	BAL	AL	6	6	0	0	0	0	0	0	0	0	0	2	0	0
95192	zwilldu01	1910	1	CHA	AL	27	27	87	7	16	5	0	0	5	1		11		
95193	zwilldu01	1914	1	CHF	FL	154	154	592	91	185	38	8	16	95	21		46	68	
95194	zwilldu01	1915	1	CHF	FL	150	150	548	65	157	32	7	13	94	24		67	65	
95195	zwilldu01	1916	1	CHN	NL	35	35	53	4	6	1	0	1	8	0		4	6	
95196																			

Batting

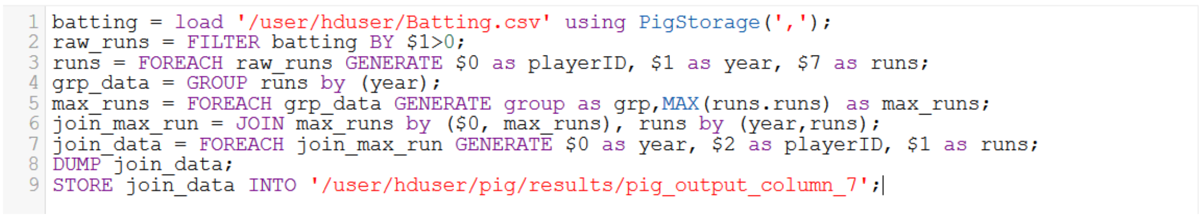
Figure 6.1.1: Data file to be processed.

The figure depicts the data included in the Batting.csv file. In order to process this file we need to upload it from our local machine to the HDFS. To achieve that, we run the command "hadoop fs –copyFromLocal Batting.csv /user/hduser" which will copy the data to the folder /user/hduser on HDFS.

The next step is to write the Pig scripts that will process this data file and produce some results. Our experiment will be a workflow composed of four pig scripts. The first and the last script's purpose, is to print in a file, the start and the finish time of the workflow, in milliseconds, so as to record the duration of the procedure in a four node cluster. The Pig command for the current time is to run in the Pig terminal editor "grunt> CurrentTime()". Although, getting the current time from a script on the HDFS we need to do a workaround. We create a text file on the HDFS and we write anything inside, such as "placeholder". In our case we created a "random_text.txt" file. Then, the Pig script will load the text file and we will ask the current time during a dummy procedure running on the file. Below, we can see the script:

```
random_text = load '/user/hduser/random_text.txt' using PigStorage(',');
current_start_time_data = foreach random_text generate CurrentTime();
STORE                      current_start_time_data                      INTO
'/user/hduser/pig/results/start_time.txt';
```

The last line commands to save the result of the CurrentTime() to the "start_time.txt" file, which is in the “/user/hduser/pig/results” directory in the HDFS. The same script will be used for the end of the workflow, but the last line’s destination will be replaced by the “end_time.txt” file, where the finish time will be recorded. The first Pig script that will be created for the processing of the uploaded data file is represented below.



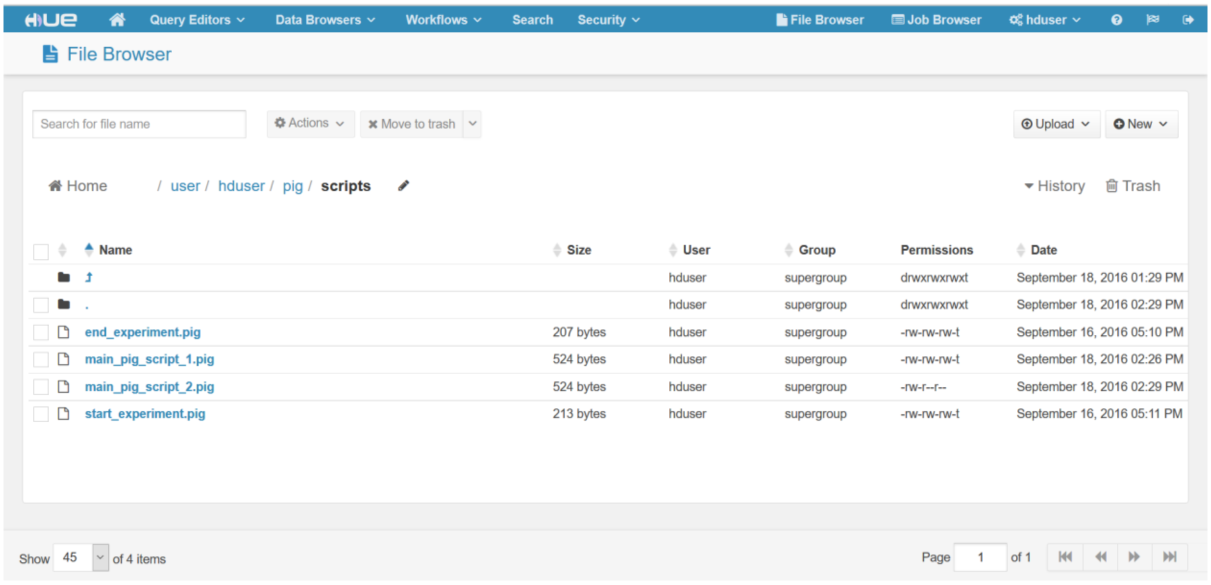
```
batting = load '/user/hduser/Batting.csv' using PigStorage(',');
raw_runs = FILTER batting BY $1>0;
runs = FOREACH raw_runs GENERATE $0 as playerID, $1 as year, $7 as runs;
grp_data = GROUP runs by (year);
max_runs = FOREACH grp_data GENERATE group as grp,MAX(runs.runs) as max_runs;
join_max_run = JOIN max_runs by ($0, max_runs), runs by (year,runs);
join_data = FOREACH join_max_run GENERATE $0 as year, $2 as playerID, $1 as runs;
DUMP join_data;
STORE join_data INTO '/user/hduser/pig/results/pig_output_column_7';
```

Figure 6.1.2: Pig script.

The first line loads the Batting.csv file in the batting variable. Then, we filter out the first row of the data. Afterwards, for every raw, we generate a description name for every column and more specifically, we name the first column as playerID, the second as year and the eighth as runs. In the next line we group the data by the years. Then, the max “runs” values are extracted and finally, we produce an output with three columns per year. The first column holds the year, the second holds the playerID and the third the runs. The output is going to be saved in the “pig_output_column_7” file in the “/user/hduser/pig/results” directory on HDFS. [21]

The same script with some differentiations will be our second Pig script, which will process the Batting.csv file. We will follow the same procedure, but in order to extract the best players according to the runs from the ninth column. So, the only changes will happen to the third line of the script, by replacing the “$7” with “$8” and the last line, so as to change the destination file to “pig_output_column_8”.

The scripts should be saved as <script name>.pig in the HDFS.



	Name	Size	User	Group	Permissions	Date
	↥		hduser	supergroup	drwxrwxrwxt	September 18, 2016 01:29 PM
	.		hduser	supergroup	drwxrwxrwxt	September 18, 2016 02:29 PM
	end_experiment.pig	207 bytes	hduser	supergroup	-rw-rw-rw-t	September 16, 2016 05:10 PM
	main_pig_script_1.pig	524 bytes	hduser	supergroup	-rw-rw-rw-t	September 18, 2016 02:26 PM
	main_pig_script_2.pig	524 bytes	hduser	supergroup	-rw-r--r--	September 18, 2016 02:29 PM
	start_experiment.pig	213 bytes	hduser	supergroup	-rw-rw-rw-t	September 16, 2016 05:11 PM

Figure 6.1.3: Pig scripts on HDFS.

We are going to create a workflow with the Pig scripts displayed above. As a start, we visit the Hue Hadoop GUI, by typing in the browser the Name Node's IP to the PORT 8888. After the login, we press the "Workflows" dropdown menu and we choose "Editors". In this screen we press the "create" button and we actually create an oozie workflow. Initially, we add a title and a description in our workflow, and then we drag and drop the Pig icon in the workflow editor. There will appear a prompt window that will ask for the path in which the pig script is saved, in the HDFS.

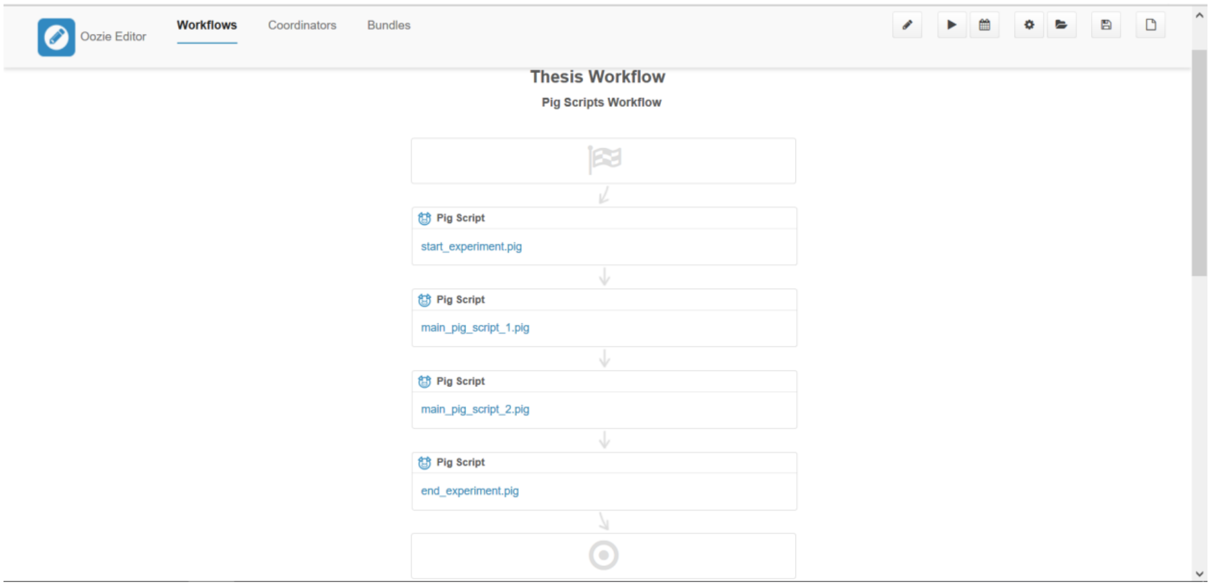
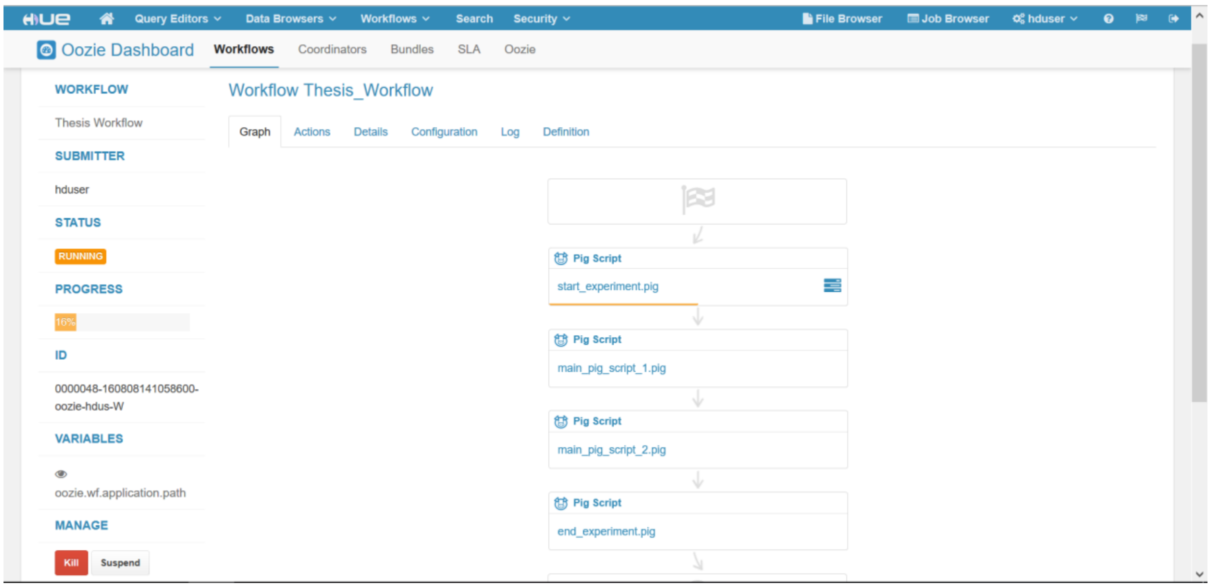


Figure 6.1.4: Oozie workflow creation with Hue.

After loading all the Pig scripts that we need to run and putting them to the correct order, we save the workflow and run it, by pressing the play button. When the workflow starts the execution, we will redirect to the screen depicted below.



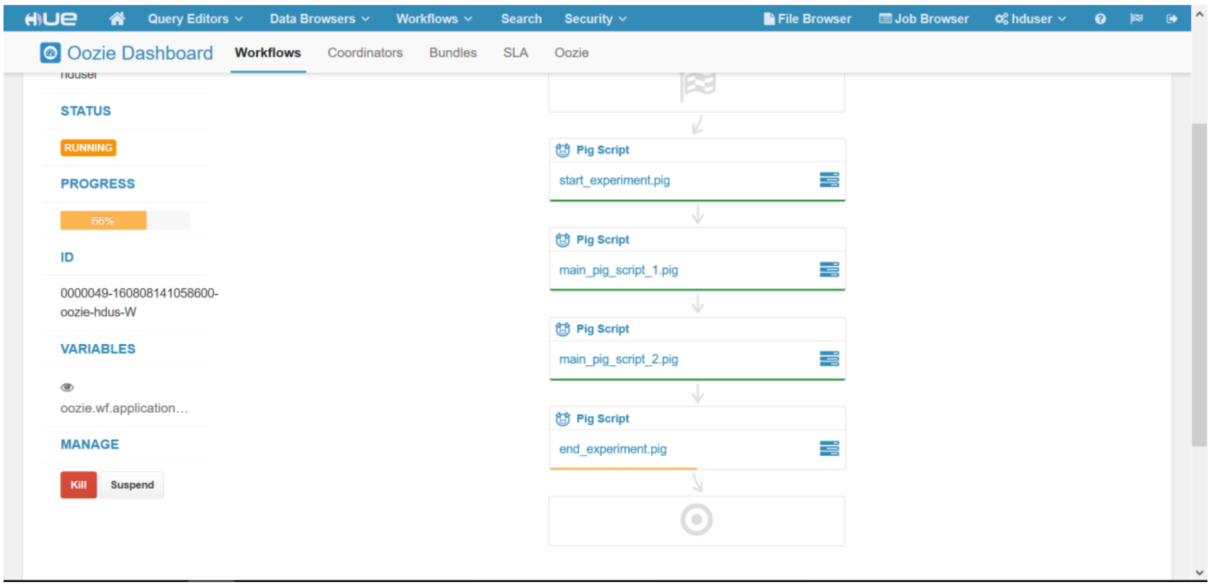
Workflow Thesis_Workflow

WORKFLOW
Thesis Workflow
SUBMITTER
hduser
STATUS
RUNNING
PROGRESS
16%
ID
0000048-160808141058600-oozie-hdus-W
VARIABLES
oozie.wf.application.path
MANAGE

Pig Script
start_experiment.pig
Pig Script
main_pig_script_1.pig
Pig Script
main_pig_script_2.pig
Pig Script
end_experiment.pig

Figure 6.1.5: Workflow running initial phase.

In this screen we can see the progress of the workflow job execution. The first script is the first job to be executed and then the procedure will continue.

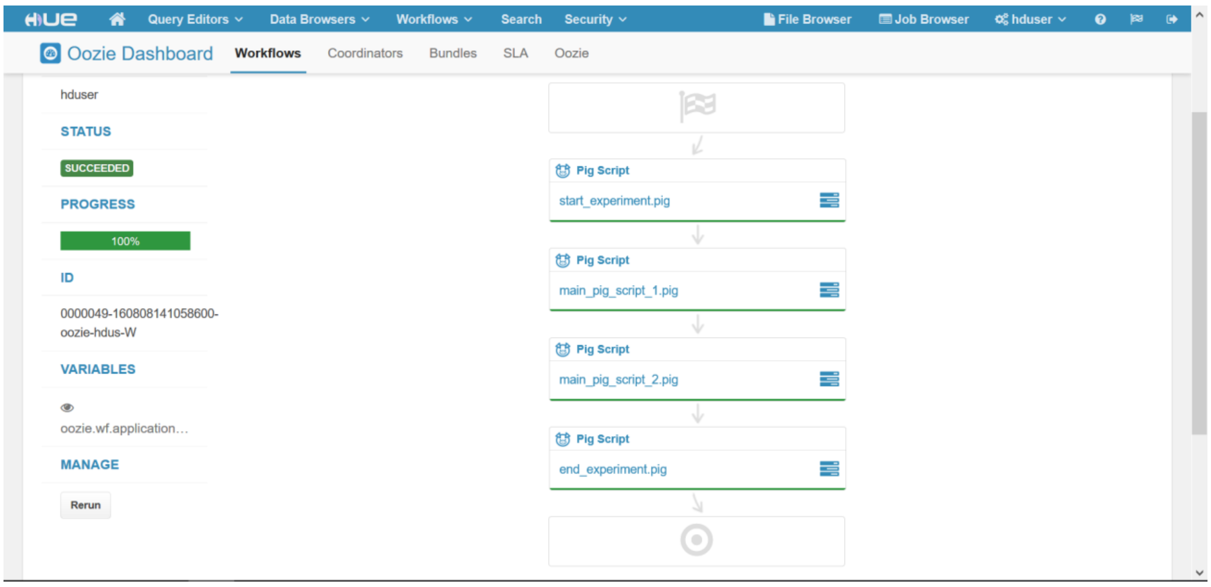


hduser
STATUS
RUNNING
PROGRESS
66%
ID
0000049-160808141058600-oozie-hdus-W
VARIABLES
oozie.wf.application...
MANAGE

Pig Script
start_experiment.pig
Pig Script
main_pig_script_1.pig
Pig Script
main_pig_script_2.pig
Pig Script
end_experiment.pig

Figure 6.1.6: Workflow running prefinished phase

When a job is finished successfully, it is coloured with green. In case of a failure we would see red. The execution of the next Pig script is an automated procedure.



hduser

STATUS

SUCCEEDED

PROGRESS

100%

ID

0000049-160808141058600-oozie-hdus-W

VARIABLES

oozie.wf.application…

Pig Script
start_experiment.pig

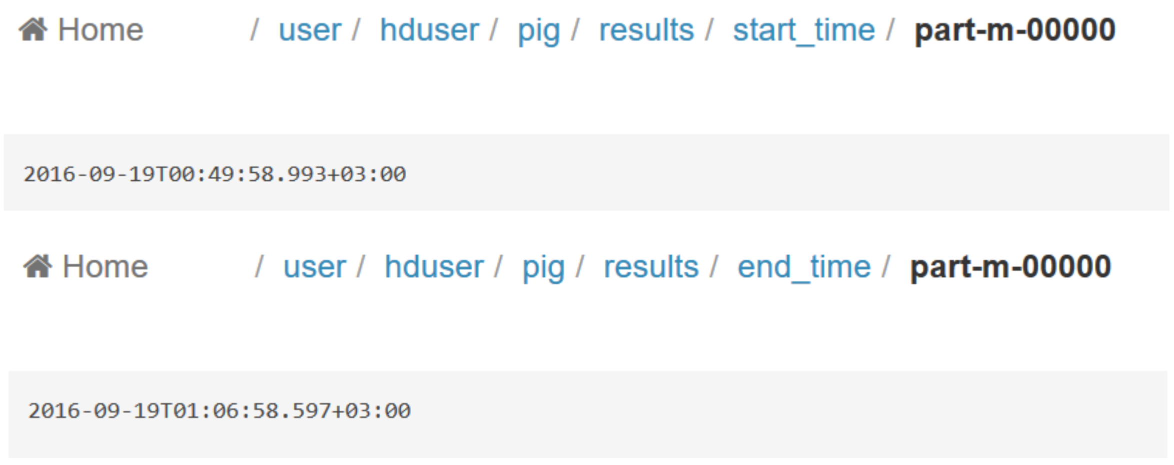
Pig Script
main_pig_script_1.pig

Pig Script
main_pig_script_2.pig

Pig Script
end_experiment.pig

Figure 6.1.7: Workflow finished with success.

This is the result produced when the workflow has succeeded. The status is success and the progress has reached the 100%. We now visit the files produced from the execution of the workflow. Firstly, we can see the "/user/hduser/pig/results/end_time/part-m-00000" and the same in the directory "start_time", in order to see the timestamps and derive information about the duration of the execution in the current four node Hadoop cluster. The files contents are displayed below.



Home / user / hduser / pig / results / start_time / part-m-00000

```
2016-09-19T00:49:58.993+03:00
```

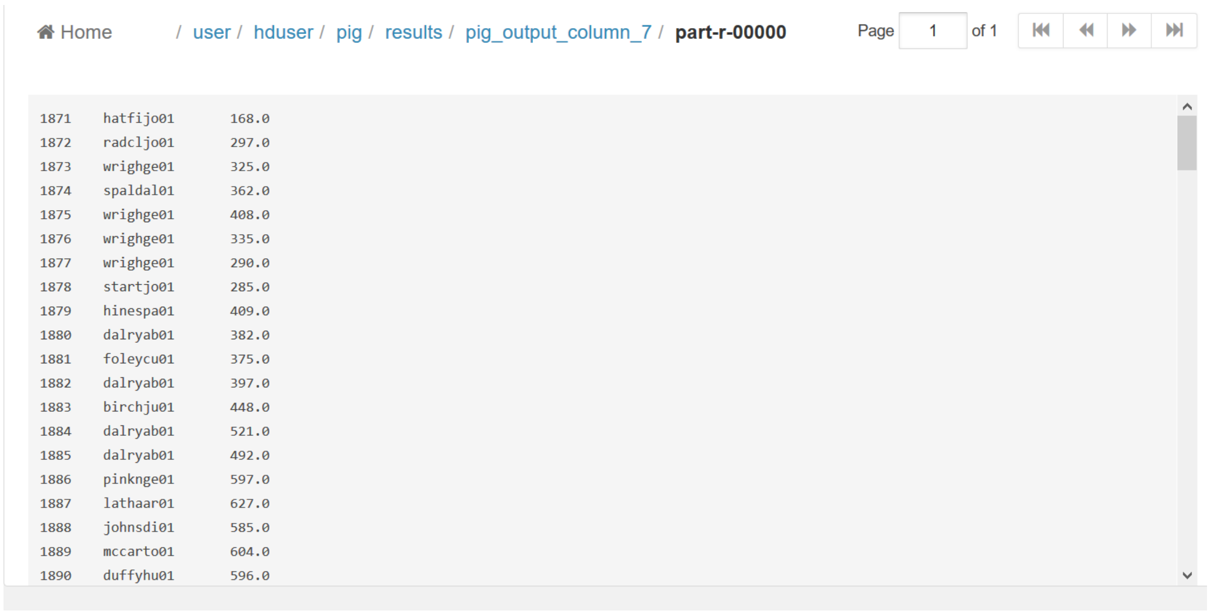
Home / user / hduser / pig / results / end_time / part-m-00000

```
2016-09-19T01:06:58.597+03:00
```

Figure 6.1.8: Workflow duration in a four node Hadoop cluster.

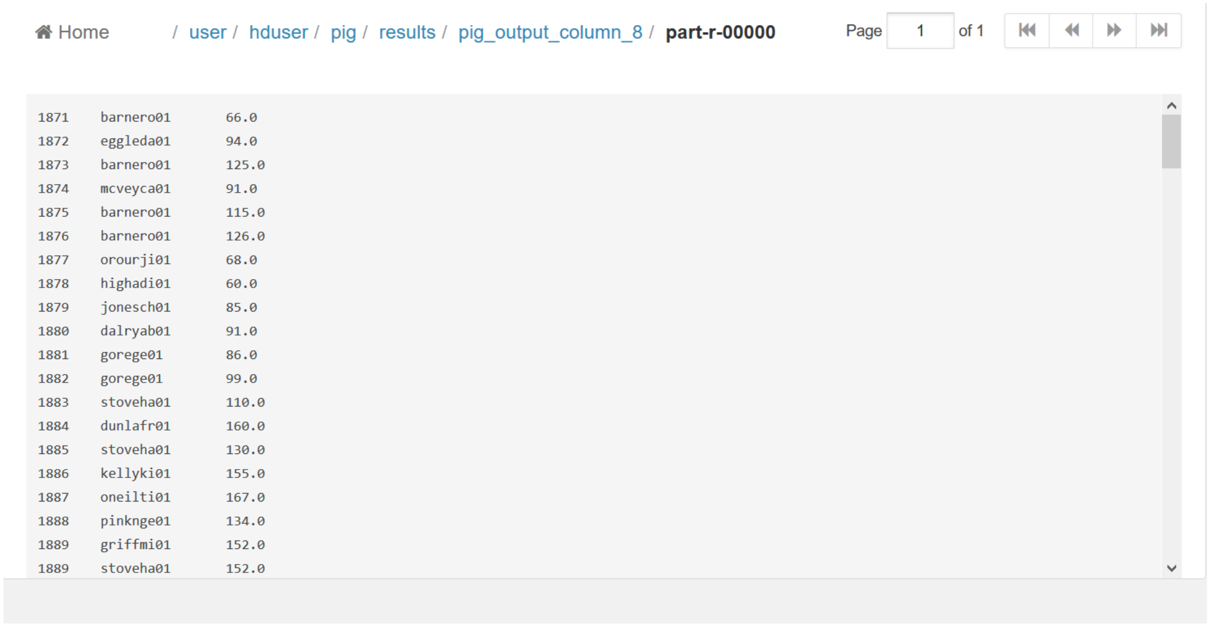
From the timestamps recorded during the procedure, we calculate that the workflow execution required **17 minutes** to be completed, in a **four node** Hadoop cluster.

The result produced by the execution of the other two scripts, "main_pig_script_1.pig" and “main_pig_script_2.pig”, is depicted below.



/ user / hduser / pig / results / pig_output_column_7 / part-r-00000

```
1871	hatfijo01	168.0
1872	radcljo01	297.0
1873	wrighge01	325.0
1874	spaldal01	362.0
1875	wrighge01	408.0
1876	wrighge01	335.0
1877	wrighge01	290.0
1878	startjo01	285.0
1879	hinespa01	409.0
1880	dalryab01	382.0
1881	foleycu01	375.0
1882	dalryab01	397.0
1883	birchju01	448.0
1884	dalryab01	521.0
1885	dalryab01	492.0
1886	pinknge01	597.0
1887	lathaar01	627.0
1888	johnsdi01	585.0
1889	mccarto01	604.0
1890	duffyhu01	596.0
```



/ user / hduser / pig / results / pig_output_column_8 / part-r-00000

```
1871	barnero01	66.0
1872	eggleda01	94.0
1873	barnero01	125.0
1874	mcveyca01	91.0
1875	barnero01	115.0
1876	barnero01	126.0
1877	orourji01	68.0
1878	highadi01	60.0
1879	jonesch01	85.0
1880	dalryab01	91.0
1881	gorege01	86.0
1882	gorege01	99.0
1883	stoveha01	110.0
1884	dunlafr01	160.0
1885	stoveha01	130.0
1886	kellyki01	155.0
1887	oneilti01	167.0
1888	pinknge01	134.0
1889	griffmi01	152.0
1889	stoveha01	152.0
```

Figure 6.1.9: Pig scripts processing results.

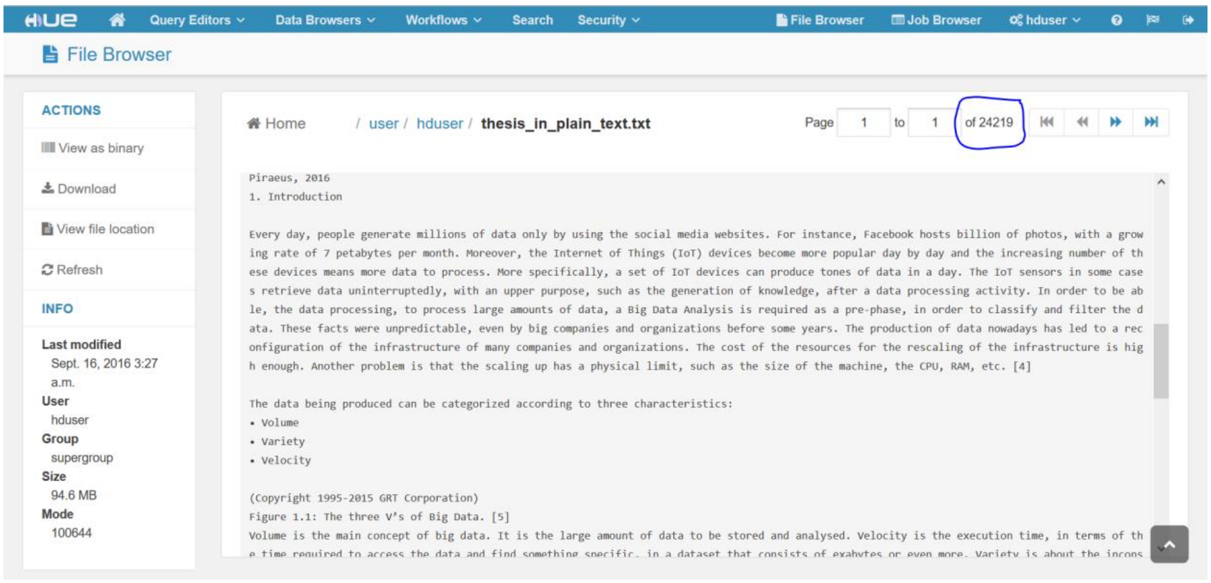
These two scripts have processed the Batting.csv file and have produced the above output, which lists by year the best player, with player’s ID, in terms of best score (depicted in the third column), considering the given input, which in the first script was the eighth column and in the second the ninth.

6.1.1 Running Map Reduce

In order to compare the duration of a job running in a four node cluster to a job running in a three node cluster we will also perform a Map Reduce action. To achieve that combined with a print of a timestamp before and after the procedure, so as to calculate the duration of the job, we will create a Linux shell script. The script will contain the next lines:

```
#!/bin/bash
date
hadoop    jar    /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.5.2.jar      wordcount      /user/hduser/thesis_in_plain_text.txt
/user/hduser/thesis_wordcount_output
date
echo "SUCCESS"
```

The first line indicates that this is a shell script. The “date” command produces a timestamp printed in the terminal screen. The hadoop jar command is a built-in example algorithm of Map Reduce offered by Apache Hadoop. In this command we specify that we will do a wordcount to the file “thesis_in_plain_text.txt” and the output will be saved in the folder “thesis_wordcount_output”. The input data (thesis_in_plain_text.txt) is a copy of this master thesis in plain text and replicated many times, until the lines reached a million lines. The purpose was to produce a huge amount of words, which is difficult to process without Hadoop.

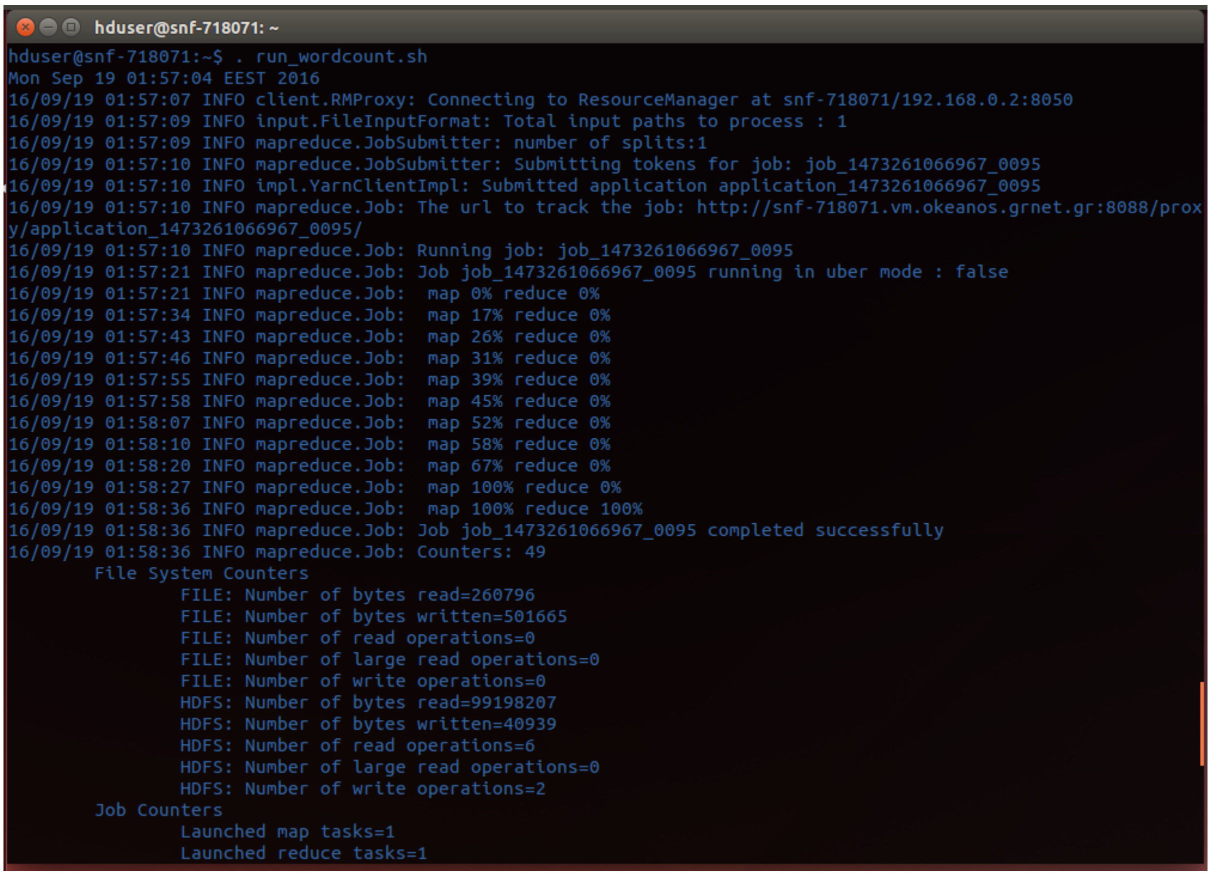


Last modified
Sept. 16, 2016 3:27 a.m.
User
hduser
Group
supergroup
Size
94.6 MB
Mode
100644

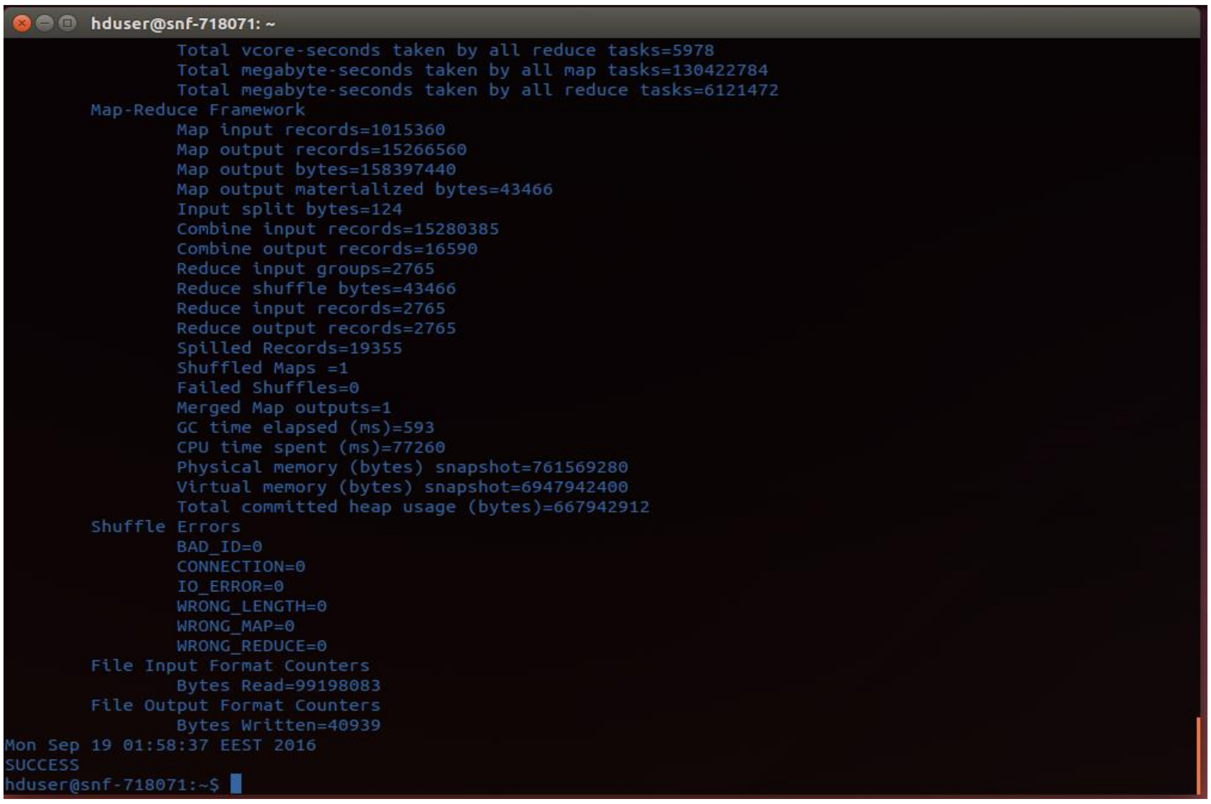
```
Piraeus, 2016
1. Introduction

Every day, people generate millions of data only by using the social media websites. For instance, Facebook hosts billion of photos, with a grow
ing rate of 7 petabytes per month. Moreover, the Internet of Things (IoT) devices become more popular day by day and the increasing number of th
ese devices means more data to process. More specifically, a set of IoT devices can produce tones of data in a day. The IoT sensors in some case
s retrieve data uninterruptedly, with an upper purpose, such as the generation of knowledge, after a data processing activity. In order to be ab
le, the data processing, to process large amounts of data, a Big Data Analysis is required as a pre-phase, in order to classify and filter the d
ata. These facts were unpredictable, even by big companies and organizations before some years. The production of data nowadays has led to a rec
onfiguration of the infrastructure of many companies and organizations. The cost of the resources for the rescaling of the infrastructure is hig
h enough. Another problem is that the scaling up has a physical limit, such as the size of the machine, the CPU, RAM, etc. [4]

The data being produced can be categorized according to three characteristics:
• Volume
• Variety
• Velocity

(Copyright 1995-2015 GRT Corporation)
Figure 1.1: The three V's of Big Data. [5]
Volume is the main concept of big data. It is the large amount of data to be stored and analysed. Velocity is the execution time, in terms of th
e time required to access the data and find something specific, in a dataset that consists of exabytes or even more. Variety is about the incons
```

Figure 6.1.1.1: Input data for Map Reduce in a four node Hadoop cluster.

The file is uploaded in the HDFS, the same way as the Batting.csv file in the previous subsection. Finally, the last line of the Linux script prints in the terminal screen the word "SUCCESS", indicating that the process has finished. The script is stored with the name "run_wordcount.sh" and can be executed by running ". run_wordcount.sh".



```
hduser@snf-718071:~$ . run_wordcount.sh
Mon Sep 19 01:57:04 EEST 2016
16/09/19 01:57:07 INFO client.RMProxy: Connecting to ResourceManager at snf-718071/192.168.0.2:8050
16/09/19 01:57:09 INFO input.FileInputFormat: Total input paths to process : 1
16/09/19 01:57:09 INFO mapreduce.JobSubmitter: number of splits:1
16/09/19 01:57:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1473261066967_0095
16/09/19 01:57:10 INFO impl.YarnClientImpl: Submitted application application_1473261066967_0095
16/09/19 01:57:10 INFO mapreduce.Job: The url to track the job: http://snf-718071.vm.okeanos.grnet.gr:8088/prox
y/application_1473261066967_0095/
16/09/19 01:57:10 INFO mapreduce.Job: Running job: job_1473261066967_0095
16/09/19 01:57:21 INFO mapreduce.Job: Job job_1473261066967_0095 running in uber mode : false
16/09/19 01:57:21 INFO mapreduce.Job:  map 0% reduce 0%
16/09/19 01:57:34 INFO mapreduce.Job:  map 17% reduce 0%
16/09/19 01:57:43 INFO mapreduce.Job:  map 26% reduce 0%
16/09/19 01:57:46 INFO mapreduce.Job:  map 31% reduce 0%
16/09/19 01:57:55 INFO mapreduce.Job:  map 39% reduce 0%
16/09/19 01:57:58 INFO mapreduce.Job:  map 45% reduce 0%
16/09/19 01:58:07 INFO mapreduce.Job:  map 52% reduce 0%
16/09/19 01:58:10 INFO mapreduce.Job:  map 58% reduce 0%
16/09/19 01:58:20 INFO mapreduce.Job:  map 67% reduce 0%
16/09/19 01:58:27 INFO mapreduce.Job:  map 100% reduce 0%
16/09/19 01:58:36 INFO mapreduce.Job:  map 100% reduce 100%
16/09/19 01:58:36 INFO mapreduce.Job: Job job_1473261066967_0095 completed successfully
16/09/19 01:58:36 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=260796
                FILE: Number of bytes written=501665
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=99198207
                HDFS: Number of bytes written=40939
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
```



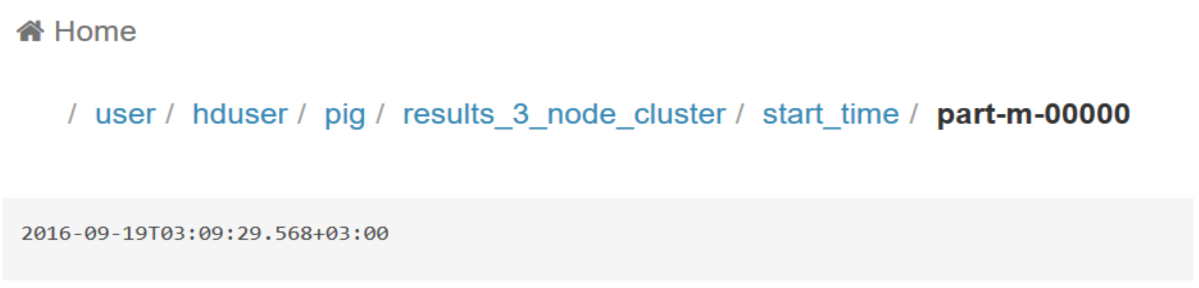
```
                Total vcore-seconds taken by all reduce tasks=5978
                Total megabyte-seconds taken by all map tasks=130422784
                Total megabyte-seconds taken by all reduce tasks=6121472
        Map-Reduce Framework
                Map input records=1015360
                Map output records=15266560
                Map output bytes=158397440
                Map output materialized bytes=43466
                Input split bytes=124
                Combine input records=15280385
                Combine output records=16590
                Reduce input groups=2765
                Reduce shuffle bytes=43466
                Reduce input records=2765
                Reduce output records=2765
                Spilled Records=19355
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=593
                CPU time spent (ms)=77260
                Physical memory (bytes) snapshot=761569280
                Virtual memory (bytes) snapshot=6947942400
                Total committed heap usage (bytes)=667942912
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=99198083
        File Output Format Counters
                Bytes Written=40939
Mon Sep 19 01:58:37 EEST 2016
SUCCESS
hduser@snf-718071:~$
```

Figure 6.1.1.2: Map Reduce job in a four node Hadoop cluster.

In the above figure it is depicted the output from the execution of the Map Reduce job. We can see the mapping procedure more analytically with percentage during the action, as it requires more time to complete, and the reduce part, which completes very fast. The amount of time required for the execution of the job can be found by subtracting the finish timestamp, which is displayed above the SUCCESS line in the terminal screen of the figure and the start time, which is in the first line. So, the execution time of the Map Reduce job was **1 minute and 33 seconds**, in a **four node** Hadoop cluster.

6.2 Evaluation – Results on a three node cluster

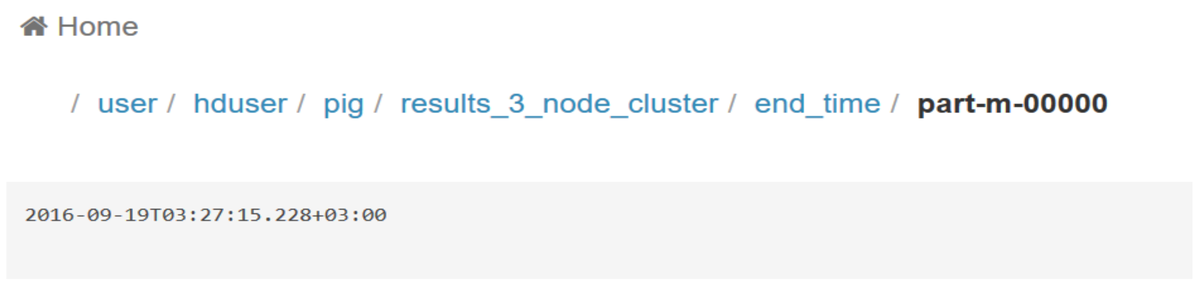
In this section we remove a node from the four node Hadoop cluster that we used previously, so now we will have a Name Node with two Data Nodes. The node removal procedure is explained in the chapter 4, section 4.2. In order to compare the previous results with those that are going to be produced, we will use the same input data file (Batting.csv) as in the previous section, as well as the same Pig scripts. Following the same method as previously we open the Hue's oozie editor and we find the saved workflow, which includes our Pig scripts. We run the workflow and we wait until it is finished. The result will be the same as before, but the files in the folders "/user/hduser/pig/results/end_time" and "/user/hduser/pig/results/start_time" will contain different timestamps.



Home

/ user / hduser / pig / results_3_node_cluster / start_time / **part-m-00000**

```
2016-09-19T03:09:29.568+03:00
```



Home

/ user / hduser / pig / results_3_node_cluster / end_time / **part-m-00000**

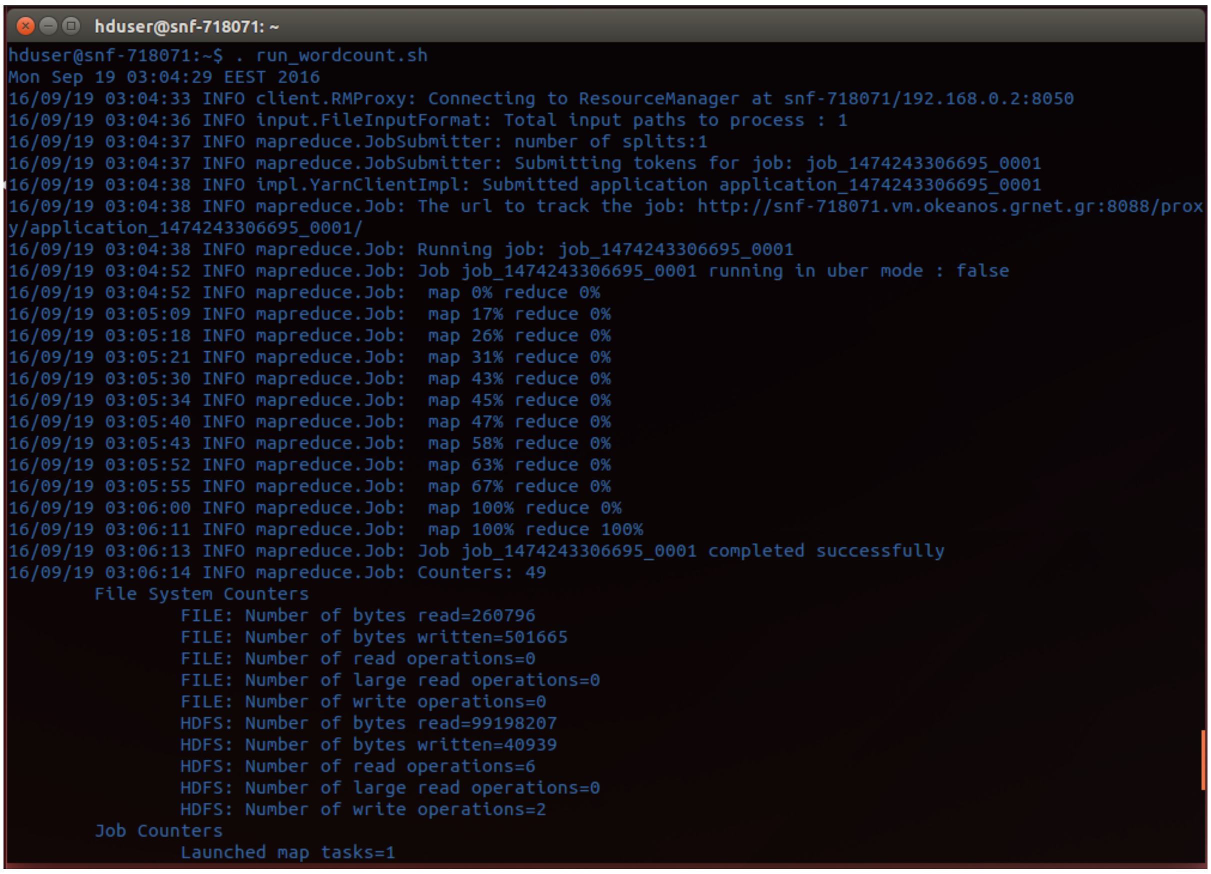
```
2016-09-19T03:27:15.228+03:00
```

Figure 6.2.1: Workflow duration in a three node Hadoop cluster.

From the timestamps recorded during the procedure, we calculate that the workflow execution required **17 minutes and 46 seconds** to be completed, in a **three node** Hadoop cluster, which is 46 seconds more than previously. We can conclude that an extra node in a Hadoop cluster, offers velocity except for storage, thanks to its processing power. Of course, in order to achieve a remarkable result it is necessary to create a larger scale cluster, but this is not inevitable for big companies that will actually need to process great amounts of data to offer an application or contract an analysis.

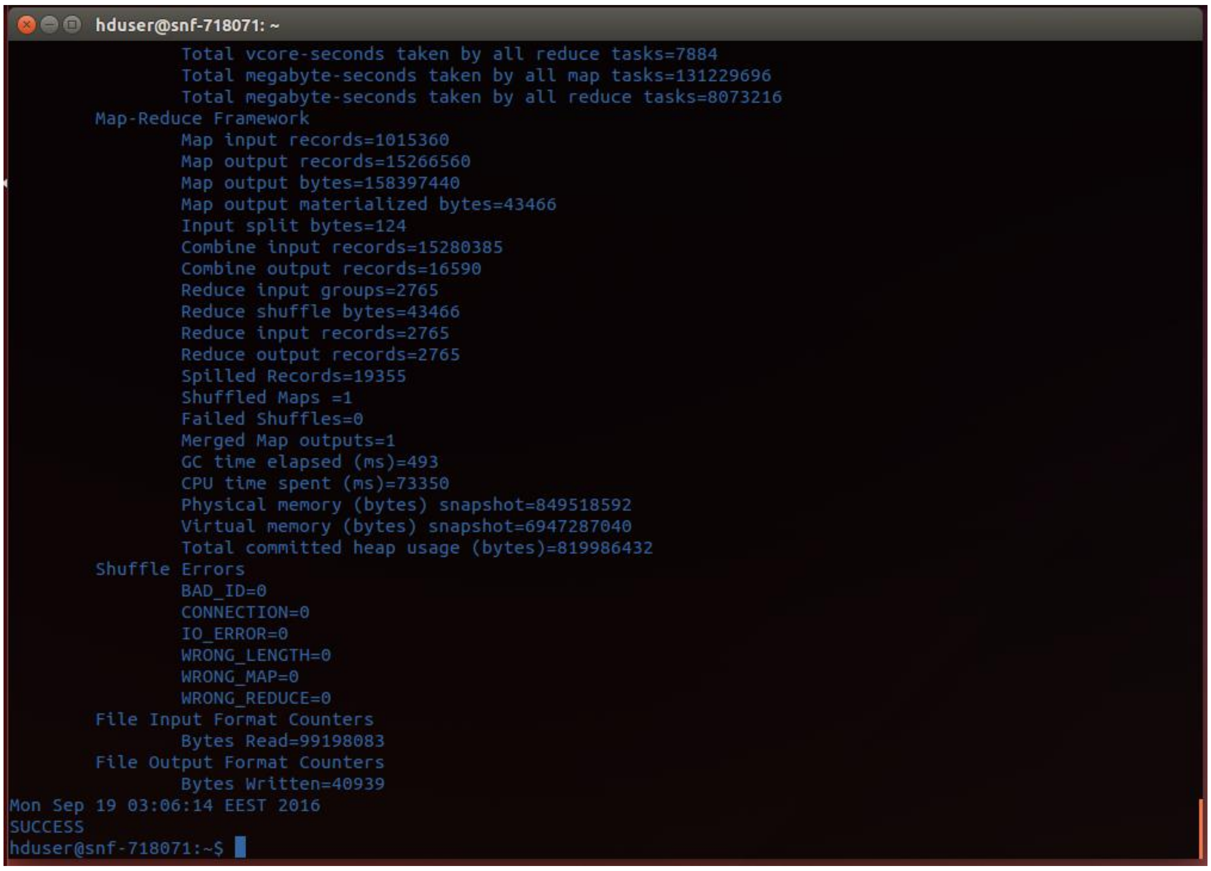
6.2.1 Running Map Reduce

The Map Reduce job will run again in the “thesis_in_plain_text.txt” file, by using the script “run_wordcount.sh”, so as to compare the execution time, in the three node Hadoop cluster, with the previous one.



hduser@snf-718071: ~

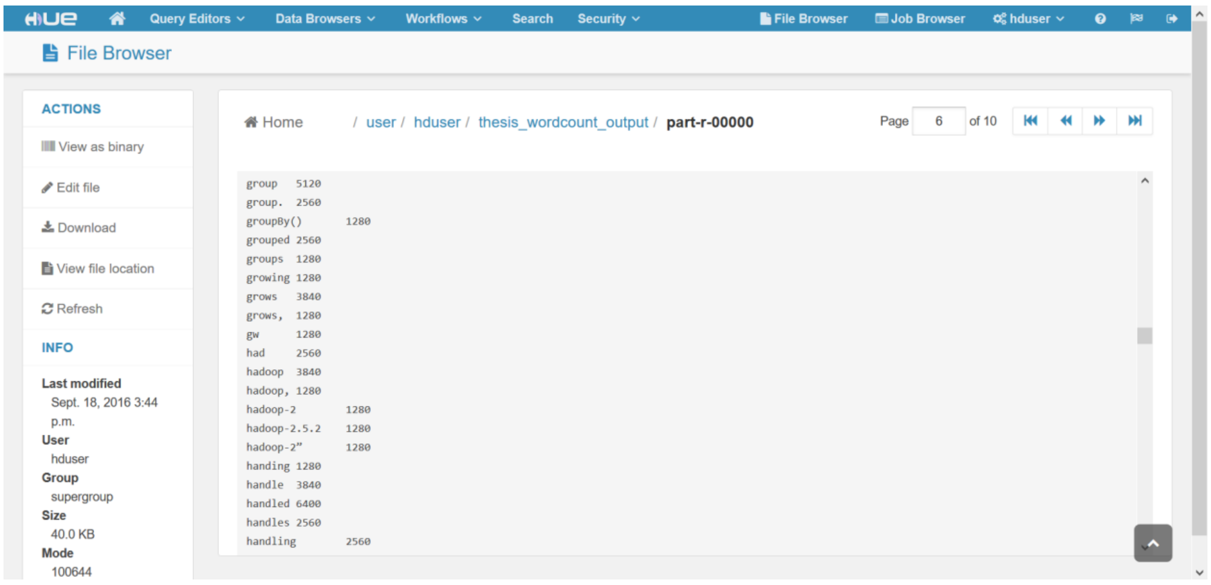
```
hduser@snf-718071:~$ . run_wordcount.sh
Mon Sep 19 03:04:29 EEST 2016
16/09/19 03:04:33 INFO client.RMProxy: Connecting to ResourceManager at snf-718071/192.168.0.2:8050
16/09/19 03:04:36 INFO input.FileInputFormat: Total input paths to process : 1
16/09/19 03:04:37 INFO mapreduce.JobSubmitter: number of splits:1
16/09/19 03:04:37 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1474243306695_0001
16/09/19 03:04:38 INFO impl.YarnClientImpl: Submitted application application_1474243306695_0001
16/09/19 03:04:38 INFO mapreduce.Job: The url to track the job: http://snf-718071.vm.okeanos.grnet.gr:8088/prox
y/application_1474243306695_0001/
16/09/19 03:04:38 INFO mapreduce.Job: Running job: job_1474243306695_0001
16/09/19 03:04:52 INFO mapreduce.Job: Job job_1474243306695_0001 running in uber mode : false
16/09/19 03:04:52 INFO mapreduce.Job:  map 0% reduce 0%
16/09/19 03:05:09 INFO mapreduce.Job:  map 17% reduce 0%
16/09/19 03:05:18 INFO mapreduce.Job:  map 26% reduce 0%
16/09/19 03:05:21 INFO mapreduce.Job:  map 31% reduce 0%
16/09/19 03:05:30 INFO mapreduce.Job:  map 43% reduce 0%
16/09/19 03:05:34 INFO mapreduce.Job:  map 45% reduce 0%
16/09/19 03:05:40 INFO mapreduce.Job:  map 47% reduce 0%
16/09/19 03:05:43 INFO mapreduce.Job:  map 58% reduce 0%
16/09/19 03:05:52 INFO mapreduce.Job:  map 63% reduce 0%
16/09/19 03:05:55 INFO mapreduce.Job:  map 67% reduce 0%
16/09/19 03:06:00 INFO mapreduce.Job:  map 100% reduce 0%
16/09/19 03:06:11 INFO mapreduce.Job:  map 100% reduce 100%
16/09/19 03:06:13 INFO mapreduce.Job: Job job_1474243306695_0001 completed successfully
16/09/19 03:06:14 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=260796
                FILE: Number of bytes written=501665
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=99198207
                HDFS: Number of bytes written=40939
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
```



```
hduser@snf-718071: ~
                Total vcore-seconds taken by all reduce tasks=7884
                Total megabyte-seconds taken by all map tasks=131229696
                Total megabyte-seconds taken by all reduce tasks=8073216
        Map-Reduce Framework
                Map input records=1015360
                Map output records=15266560
                Map output bytes=158397440
                Map output materialized bytes=43466
                Input split bytes=124
                Combine input records=15280385
                Combine output records=16590
                Reduce input groups=2765
                Reduce shuffle bytes=43466
                Reduce input records=2765
                Reduce output records=2765
                Spilled Records=19355
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=493
                CPU time spent (ms)=73350
                Physical memory (bytes) snapshot=849518592
                Virtual memory (bytes) snapshot=6947287040
                Total committed heap usage (bytes)=819986432
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=99198083
        File Output Format Counters
                Bytes Written=40939
Mon Sep 19 03:06:14 EEST 2016
SUCCESS
hduser@snf-718071:~$ 
```

Figure 6.2.1.1: Map Reduce job in a three node Hadoop cluster.

In the above figure it is depicted the output from the execution of the Map Reduce job in a three node Hadoop cluster. The amount of time required for the execution of the job can be calculated considering the finish timestamp, which is displayed above the SUCCESS line in the terminal screen of the figure and the start time, which is in the first line. So, the execution time of the Map Reduce job was **1 minutes and 45 seconds**, in a **three node** Hadoop cluster, which is 12 seconds more than in the previous case. A sample of the output produced by the word count job is displayed in the next figure.



Last modified
Sept. 18, 2016 3:44 p.m.
User
hduser
Group
supergroup
Size
40.0 KB
Mode
100644

Home / user / hduser / thesis_wordcount_output / part-r-00000

Page 6 of 10

```
group   5120
group.  2560
groupBy()       1280
grouped 2560
groups  1280
growing 1280
grows   3840
grows,  1280
gw      1280
had     2560
hadoop  3840
hadoop, 1280
hadoop-2        1280
hadoop-2.5.2    1280
hadoop-2”       1280
handing 1280
handle  3840
handled 6400
handles 2560
handling        2560
```

Figure 6.2.1.2: Map Reduce sample output.

The file “thesis_in_plain_text.txt”, on which we run the Map Reduce word count job, is a copy of this master thesis, replicated many times until reaching a million lines. That is the reason why the words are appeared thousand times.

7. Conclusions

From the experiments conducted during the composition of this master thesis, we conclude that the Apache Hadoop is a powerful framework for the processing and analysis of huge amounts of data. The data varies from scientific data, which could be measurements from sensors for any kind of science that needs to analyse occurrences and behaviours, to every day data, which could be taxes data for every person of a country per year or even football data, which holds the players performance and someone may need to run a statistical analysis on that dataset to find the best world player per year, according to a specific filter, such as the greatest number of goals.

There are many components that can be combined with the Hadoop framework and create an enriched Hadoop ecosystem, such as Apache Oozie, Pig, Mahout, Hive etc. All these components offer great flexibility and usage, in terms of combining different technologies with the Big Data concept. For example, Apache Mahout enables the creation and mixture of machine learning algorithms and concepts with the Hadoop. Other components, such as HBase, enable the integration of the Hadoop Distributed FileSystem with NoSQL (Not only SQL) databases. Moreover, the Apache Oozie is a great workflow scheduler that offers the ability of running many Hadoop jobs, which could be Pig scripts, Hive queries, Map Reduce jobs, etc. Also, the oozie workflows can be stored in the Hadoop Distributed FileSystem and reproduce the workflow – experiment in the future, to a larger dataset which may contain new measurements added to the previous values.

The oozie component has been used for the execution of the experiments conducted in the previous chapter. The experiments were consisted of some Pig scripts, which analysed a large dataset with a statistical purpose. The jobs of the workflow completed in 17 minutes and the results stored in the Hadoop Distributed FileSystem. The processing power of the Hadoop cluster was empowered by four machines from which the one had the role of the Name Node of the cluster and the other three were the Data Nodes.

In order to prepare a benchmarking, the workflow file, which created in the oozie editor in section 6.1, was stored in the Hadoop Distributed FileSystem for later use. According to the steps described in the section 4.2, we managed to remove a data node from the four node Hadoop cluster, so as to examine the increment of the execution time of the same workflow.

By removing the Data Node, the processing power of the cluster is reduced. The workflow jobs are executed in a distributed way, by all the Data Nodes simultaneously, so the removal of a node could have a significant delay for the execution of a job that will process a file with a size of several petabytes. In our case, the decrement of the Hadoop cluster resulted in a delay of 46 seconds in a file with a size of several megabytes (~ 6 MB). So, we can conclude that big companies and organizations are benefited from the addition of a data node in an existing Hadoop cluster, because it uses commodity hardware and offers greater distributed processing power.

As a second experiment we have run a Map Reduce job, in a file with a million lines, in order to compare the time required for the word count procedure with a four node cluster and then with a three node. The difference was only 12 seconds, but considering that the size of the file was only a few kilobytes, this means a lot. Even in a small sized file we can see a delay, when removing a node from a Hadoop cluster, which means that the processing of a large dataset would be significantly lower. Contradictory, the addition of a node is of great importance when companies have to deal with huge amounts of data within a deadline, as the procedure will complete a lot quicker.

The general picture of the Big Data Analytics Systems is that they offer flexibility and the ability to process large datasets, by using commodity hardware. Although, smaller companies with datasets that do not comprise of files greater than 100 MB, should consider to use other technologies/solutions to process their data, as they could be efficient as well. The target group of the Big Data Analytics Systems is big companies like Google that deal with enormous data files. The reason is that these systems distribute the data, which is under processing, to the data nodes and this requires some time initially. This initial delay is insignificant when dealing with large datasets, because subsequently the lost time will be gained from the parallel processing power of the data nodes. But in case of small datasets, the initial delay will be a drawback and maybe other technologies would be more effective for the analysis of the data. However, even middle companies, year by year, increase their datasets rapidly and in some years the Big Data Analytics Systems will be part of everyday life.

Bibliography – References

Books

[1] Tom White, April 2015, *Hadoop: The Definitive Guide, 4th Edition*, O'Reilly, Sebastopol, CA

[2] Sandeep Karanth, December 2014, *Mastering Hadoop: Go beyond the basics and master the next generation of Hadoop data processing platforms*, Packt Publishing Ltd., Birmingham B3 2PB, UK

[3] Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, Jeff Markham, 2014, *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*, Addison-Wesley

[6] Thilina Gunarathne, Srinath Perera, February 2015, *Hadoop MapReduce v2 Cookbook, Second Edition*, Packt Publishing Ltd., Birmingham B3 2PB, UK

[10] Rajesh Nadipalli, September 2013, *HDInsight Essentials*, Packt Publishing Ltd., Birmingham B3 2PB, UK

[11] Dirk deRoos, Jabuary 2014, *Hadoop For Dummies*, Wiley Brand / For Dummies. Also part of it available online at: http://www.dummies.com/how-to/content/developing-oozie-workflows-in-hadoop.html [Accessed: 04-Aug-16]

[14] Sameer Wadkar, Madhu Siddalingaiah, 2014, *Pro Apache Hadoop, Second Edition*, Heinz Weinheimer

Websites

[4] Understanding big data. Online Available: http://www.bigdataplanet.info/p/what-is-big-

data.html (Deepak Kumar, Big Data / Hadoop Developer, Software Engineer, Thinker, Learner, Geek, Blogger, Coder). [Accessed: 09-Jul-2016]

[5] Getting Value from Big Data – How, When and Why? Online available: http://www.grtcorp.com/solutions/data_warehouse_business_intelligence/big-data (GRT Corporation). [Accessed: 09-Jul-2016]

[7] Understanding Hadoop Ecosystem.
Online available: http://hadooptutorials.co.in/tutorials/hadoop/understanding-hadoop-ecosystem.html (by Deepesh Kumbhare, HadoopTutorials.co.in). [Accessed: 03-Aug-16]

[8] Big Data: Introduction to Hadoop. Online available: http://www.stratapps.net/intro-hadoop.php (Stratapps Inc.). [Accessed: 03-Aug-16]

[9] Introduction to Apache Hive.
Online available: http://hadooptutorials.co.in/tutorials/hive/introduction-to-apache-hive.html (by Tanmay Deshpande, HadoopTutorials.co.in). [Accessed: 04-Aug-16]

[12] New Apache Oozie Workflow, Coordinator & Bundle Editors. Online available: http://gethue.com/new-apache-oozie-workflow-coordinator-bundle-editors/ (by Hue Team, April 2015, gethue.com). [Accessed: 04-Aug-16]

[13] Hadoop 101: An Explanation of the Hadoop Ecosystem. Online available: https://dzone.com/articles/hadoop-101-explanation-hadoop (by Gil Allouche, 2014, Big Data Zone). [Accessed: 06-Aug-16]

[15] Apache Spark RDD.
Online available: http://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm (tutorialspoint.com). [Accessed: 06-Aug-16]

[16] Acquiring Big Data Using Apache Flume.
Online available: http://www.drdobbs.com/database/acquiring-big-data-using-apache-

flume/240155029 (by Dan McClary, June 2013, DrDobb's, The world of software development, drdobbs.com). [Accessed: 07-Aug-16]

[17] Hadoop Basics - Horizontal Scaling instead of Vertical Scaling. Online available: http://samjay-complete.blogspot.gr/2014/03/hadoop-basics-horizontal-scaling.html (posted by samjay, March 2014, Tech Universe). [Accessed: 08-Aug-16]

[18] Getting Started Guide – Pivotal HD.
Online available: http://pivotalhd.docs.pivotal.io/docs/getting-started.html (Pivotal Software, Inc, 2016, Pivotal Documentation). [Accessed: 22-Aug-16]

[19] Commissioning and Decommissioning of Datanode in Hadoop. Online available: https://acadgild.com/blog/commissioning-and-decommissioning-of-datanode-in-hadoop (posted by Onkar Singh, 11 March 2016, ACADGILD). [Accessed: 05-Sep-16]

[20] Hue 3 on HDP installation tutorial. Online available: http://gethue.com/hadoop-hue-3-on-hdp-installation-tutorial/ (posted by Hue Team, 12 February 2015, gethue.com). [Accessed: 13-Sep-16]

[21] How to process data with apache pig. Online available: http://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-pig/#downloading-the-data (posted by hortonworks). [Accessed: 16-Sep-16]

[22] Official Apache Oozie website, Last Published: 2016-08-17, Online available: http://oozie.apache.org/. [Accessed: 19-Sept-2016]

[23] Wikipedia , the free encyclopedia, information about Apache Oozie, Online available: https://en.wikipedia.org/wiki/Apache_Oozie. [Accessed: 19-Sept-2016]