Big Data Analysis Using Hadoop and MapReduce

Harrison Carranza, MSIS
Marist College, Harrison.Carranza2@marist.edu

Mentor: Aparicio Carranza, PhD
New York City College of Technology - CUNY, USA, acarranza@citytech.cuny.edu

Abstract*– In the last few years, the advancement of technology has been growing at a rapid rate. This trend is expected to continue to grow over the next few years due to the amount of data and information being created daily. Eventually, it is going to be difficult to manage various aspects of our information. As a result of this, it is essential to acquire the necessary tools or software to help us organize our information and use it to acquire as well as filter certain pieces of data we consider important. Specifically, Apache Hadoop is designed to perform such tasks. Its software library incorporated internally allows users to work with hundreds of nodes and petabytes of data. In addition, it allows analysis to be done among interactive queries at a very fast rate. One application in Hadoop, MapReduce, is used to filter and analyze activity log files recorded by computers at a fundamental level to help us increase awareness of the level of security in our systems and monitor any strange activities occurring in a network. With the use of the Linux operating system, we can determine the number of instances of certain activity within our system by creating commands that help parse log files without having to open up the files to manually count them.

Keywords — Big Data, Hadoop, MapReduce, Parse, Queries

I. INTRODUCTION

As the amount of data we use increases, so does the need of the analysis. In the last few decades, technology has grown vastly and will only continue to do so in the future. There was a time when personal computers were equipped with 16KB of RAM and required cassettes for storage. Today, modern computers now contain 8GB or more of RAM with internal storage of more than 1TB. This is not including external memory that could be well beyond 1TB [1].

To demonstrate how big data has evolved over time, we can consider the example of social media. As of 2014 Facebook has stored 300 Petabytes of data at a rate of 600 terabytes a day in order to handle its users' billions of photos and video [5]. YouTube users around the world upload nearly 400GB of videos online per hour. Uploading these enormous amounts of data create problems for computing power and storage capabilities. For example, 10TB of data would take 5 nodes of parallel computing power about 6 hours to process/read it whereas if this data were spread across 500 nodes it would then take around 6 minutes to read. If transferred over a network to other nodes the time would increase by some orders of magnitude [3].

Apache Hadoop is the designated software that can address these concerns by distributing the work and data over thousands of machines in clusters. The workload is spread evenly across the cluster, which allows for effective parallel computing of data. It also ensures that only a small amount of data is transferred over the network on processing time so it suits best scenarios where data is written once and gets read many times, known as WORM [1].

Hadoop consists of two parts: Hadoop Distributed File System (HDFS) and MapReduce. The former is designed to scale Petabytes of storage and runs on top of the file systems of the underlying operating system in order to provide high-throughput access to application data. The MapReduce programming model is a simple technique that allows programmers to work on an abstract level without having to play around with cluster specific details or communication and data flow. With the use of two functions, **map** and **reduce**, we are able to receive incoming data, analyze it, modify it, and then send it to an output channel.

In section I, we provided the motivation and background for this work. Section II, III, and IV present The Hadoop Framework, Map Reduce & Work Count; and Hadoop & Map Reduce, respectively. In Section V, we present our Map Reduce Experimentation. Finally, in Section VI our Conclusions are included.

II. THE HADOOP FRAMEWORK

Hadoop attempts to locate the work units on the same nodes as the data for this unit is hosted. This is important because in a data center the network bandwidth is considered to be the most precious asset if the amount of data is large. After all, Hadoop consists of the three primary resources: HDFS, MapReduce programming platform, and the Hadoop ecosystem, which is a collection of tools that organize and store data as well as manage the machines on Hadoop [2], [4].

Hadoop is written in Java but can run the MapReduce programs expressed in other languages such as Python, Ruby, and C++ [5], [6]. No matter which language is implemented in; the map function will still function properly as long as it can read and write from standard input to standard output. To understand Hadoop, it is fundamental to learn about the WordCount option using MapReduce. It is considered to be "Hello World" equivalent of Hadoop because it teaches how to begin using the software but it also provides us with the

ability to log files, analyze activity, and other advanced tasks. The main objective is going to be to perform the Word Count example making use of the MapReduce option on Hadoop in order to determine how many times a certain item or items appear in a given file. Figure 1 below shows a basic overview of Word Count in action, using letters as part of the example.

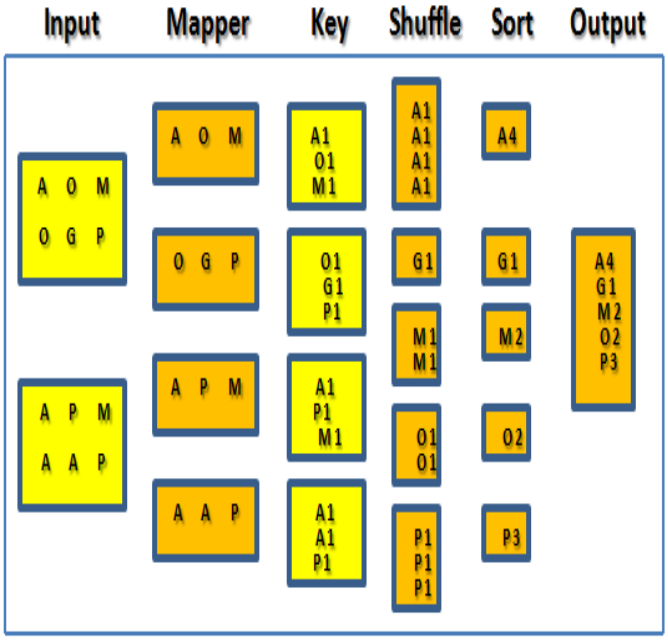

Figure 1 – Word Count Basic Overview

III. MAP REDUCE AND WORD COUNT

MapReduce was the first and is still the primary programming framework for developing applications in Hadoop. MapReduce functions with the use of Java. This application has jobs that consist of Java programs called mappers and reducers. The way that it works is that each mapper is provided data that is to be analyzed. Using a sentence for an example, we could analyze the data [2], [4].

To illustrate how MapReduce functions, we can use an example called “Word Count”. If we create a sentence, we could analyze it from top to bottom and create name-value pairs or maps. For experimentation purposes, let’s assume it gets a sentence: “My cat fell asleep on my bed.” The maps are: “my”:1, “cat”:1, “fell”:1, “asleep”:1, “on”:1, “my”:1, and “bed”:1. The name is the word, and the value is a count of how many times it appears on record. In this example, all the words would show up with a 1 next to it but “my” would be displayed as “my”:2 once the processing is done [2].

The job of Hadoop is to process all the data, in this case, the sentence, and arranges it as needed. Once the sentence is inputted into the program, the mapper analyzes it and creates a key. This key is what gathers together all the information from the mapper before they get shuffled. Once they are shuffled by the reducer, it adds up all the maps for each word, sorts them in alphabetical order with their number of occurrences, and produces the output of words listed in a text document. Essentially, mappers are programs that extract data from HDFS files into maps, and reducers as programs that take the output from the mappers and overall aggregate data [5].

IV. HADOOP AND MAPREDUCE SETUP

This experiment explores the functionality of Hadoop across certain operating systems. For our purposes, we have tested it with the 64-bit OS version of Ubuntu Linux 16.04 using a single laptop [8]. The computer used has an Intel Core i7 processor of 2.40GHz with a RAM of 6GB. There really is not a specific minimal hardware requirement considering that it requires only one system. However, it is beneficial if larger memory for the expectancy of quick and efficient results as well as for the use of more than one operating system.

The main goal of this initial part of the experiment was to perform a simple Hadoop installation in order to determine the system capabilities and build a prototype using the word count function and to learn more about how it works. In order to accomplish this, I installed Ubuntu 16 on a laptop in order to perform our primary goal of running an activity log file of a computer system that demonstrates how MapReduce functions on Hadoop.

The installation process of Hadoop on Ubuntu Linux is the first step to accomplishing this task. Since the operating system is a Linux distribution, most of the time the Java packages are already incorporated inside. However, if the programming language is not installed, there are some steps that need to be taken.

To update the package list, simply type the following command on prompt terminal of Linux:

```
$ sudo apt-get update
```

To install OpenJDK, the available Java Development Kit, type in the following:

```
$ sudo apt-get install default-jdk
```

To check the version, type in:

```
$ java –version
```

Once Java is set up on the computer, it is time to download a stable release of Hadoop. The link below can direct you to stable releases:

www.hadoop.apache.org

It is essential that the binary link is clicked on to download the most stable release. For our purposes, we used version 2.7.3 [7]. The Hadoop files should be moved to the /usr/local directory where local software is installed.

Once Hadoop is installed and configured on your computer, it is time to run it. Hadoop should be stored in the directory mentioned earlier. Figure 2 shows the command that needs to be types as well as the output.

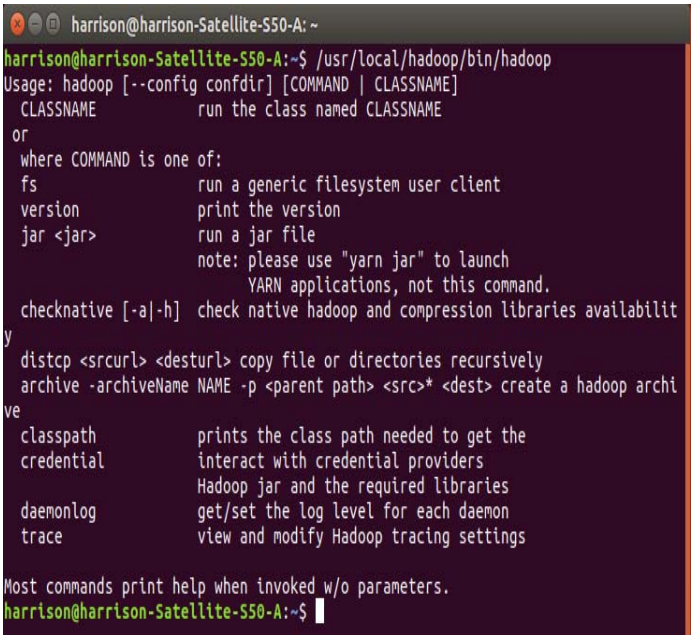

Figure 2 – Location of Hadoop Configuration Files

Once the above results are obtained, we can verify that we successfully configured Hadoop to run in stand-alone mode. To demonstrate that it is running properly, the MapReduce function is going to be put into use by typing in the following commands on the terminal:

```
$ mkdir ~/input
$ cp /usr/local/hadoop/etc/hadoop/*.xml ~/input
```

The commands above are used to show the example of MapReduce. First, we make a directory called ~/input inside our home directory to call up the necessary files later. Then we copy the configuration files from the Hadoop program to use as our data for the MapReduce example.

V. MAPREDUCE EXPERIMENTATION

Once all the files are set, it is time to experiment with the MapReduce program. The objective of the example is to determine how many times a word or phrase shows up in a file or document.

Using the grep command in Linux, MapReduce is going to count the number of instances of a literal word or phrase. For this instance, we used the word “principal” to help us prove the proper function of the program. The word “principal” is expected to show up, whether it is at the middle or end of a declarative sentence. It is important to keep in mind that the expression is case-sensitive so if we were to type in the word all in lower case, the results would not show for any word that is at the beginning of a sentence as they would start with a capital letter.

Figure 3 below shows the commands on the prompt that allows us to create the grep_example file to determine the number of instances that the word “principal” shows up across all files in the ~/input folder.

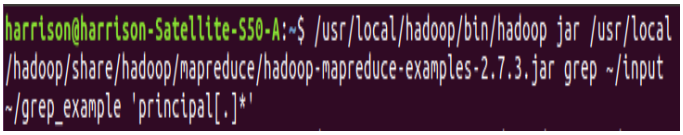

Figure 3 – Command for creating grep_example file

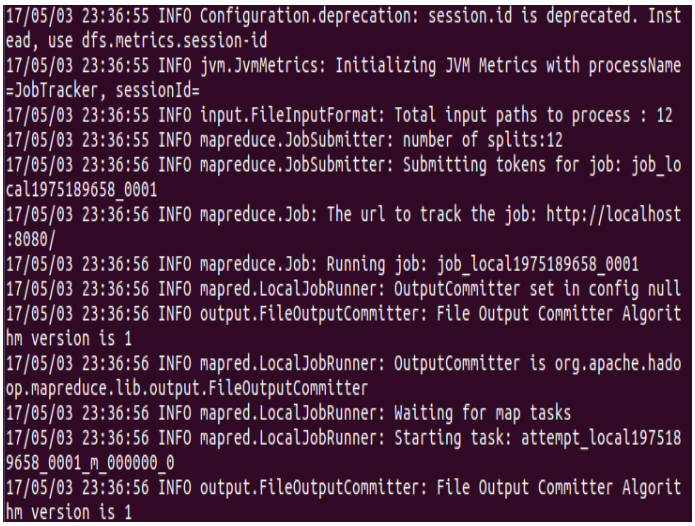

Figure 4 – Output of grep_example verifying configuration

Figure 4 above shows what the output should be once the command in the above figure is entered. It is important to know that this output shows up when the grep_example folder is being created.

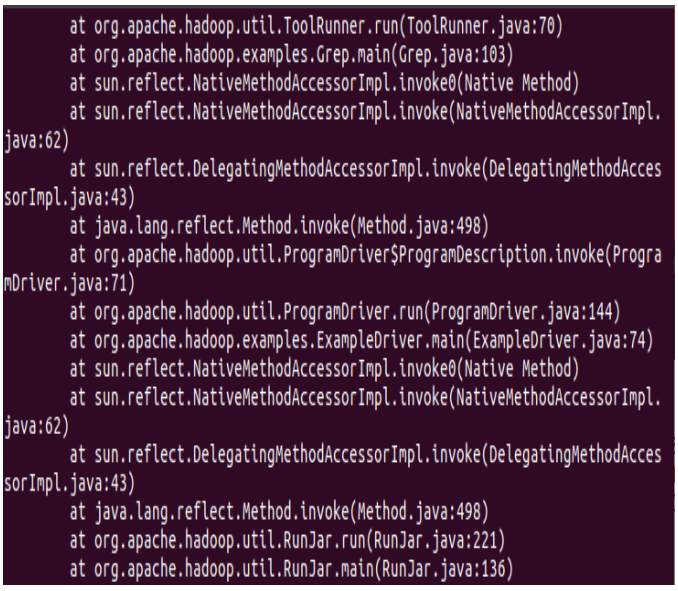

Figure 5 – Output of grep_example verifying folder existence

Figure 5 above shows another output that is produced by the command shown in the figure 2. This output shows up because the grep_example folder already exists.

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example/*
6       principal
1       principal.
```

Figure 6 – Output showing instances of “principal”

Figure 6 above shows the command that needs to be typed in to determine the necessary output. We use the concatenate command followed by the grep_example file. Once it is entered, the word “principal” shows up with the number of instances present across all files stored in the ~/input folder.

The next few screen captures are just some examples that prove the proper functioning of the MapReduce program on Hadoop using Linux. During the experimentation, we first used just one activity log file to track how many instances of access were made during a certain given period of time. However, we wanted to make sure that we could obtain consistent results, especially because eventually others shall make use of the program. In order to make that happen, we underwent some steps.

In the ~/input folder, there are several .xml files created by the installation of the Hadoop program on Linux. The activity log file given to us was placed into the input folder and then converted to an .xml file so as to make sure that it could be read by the MapReduce program. However, it is not necessary to do this since the program is going to read the archives regardless of their file extension. For this experiment, we created four identical activity log files: one in its original .txt format and the other three in .xml format.

The following data and screen captures are some examples of our investigation of MapReduce. We used five words or phrases to display how many times they are present in the input directory.

```
harrison@harrison-Satellite-S50-A:~$ /usr/local/hadoop/bin/hadoop jar /usr/local
/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep ~/input
~/grep_example1 'the[.]*'
```

Figure 7 – Command for grep_example1

In Figure 7, we run the command that helps us obtain the number of instances the word “the” is present among the files in the input folder. This word has been used since it is a common word and can help us guarantee that it shall show up several times in order to prove our point. In Figure 8 below, we can see that across all 12 files in the designated directory, the word “the” shows up 173 times. This does not consider the fact that there are times where “The” shows up with a capital letter at the beginning.

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example1/*
173     the
```

Figure 8 – Output showing instances of “the”

In the next few screen captures, we are going to focus on the amount of times certain phrases appear in the activity log files. Since we created the input folder in the previous steps, we shall use this directory to store the log files we are going to work with here. For the purposes of experimentation, we create four identical log files. The reason behind this is to demonstrate that when a user wants to search for certain instances of a word or phrase among several files, the program is going to successfully complete this task. In this situation, it is important to keep in mind that despite there being other files from the steps before, we only expect the results to originate from the four log files created in the input directory.

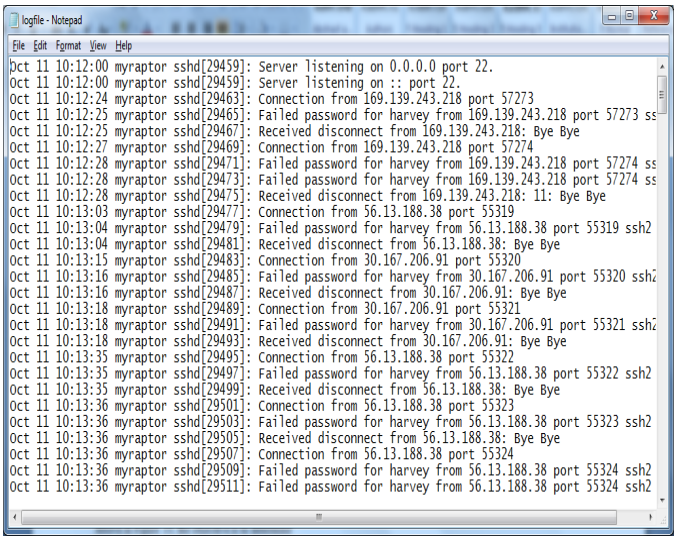

Figure 9 – Activity Log File with entries to be processed

The objective is to determine the amount of times a certain word or phrase comes up. Here, we experiment with the word “data” by typing in the command illustrated in Figure 10. Once that is executed on the Linux prompt, the output displayed in Figure 11 should show up on the screen.

```
harrison@harrison-Satellite-S50-A:~$ /usr/local/hadoop/bin/hadoop jar /usr/local
/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep ~/input
~/grep_example2 'data[.]*'
```

Figure 10 – Command for grep_example2

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example2/*
11      data
1       data.
```

Figure 11 – Output showing all instances of “data”

In Figure 11, we can see that the output is displayed on two lines. The first line indicates to us that “data” alone appears 11 times across all files in the input folder. The second line demonstrates how many times the word shows up but with a period after it, meaning that it is located at the end of a sentence.

In this next example, we shall work with a phrase instead of a word: “Server listening.” Please note that it is essential to spell it appropriately as we are dealing with a case-sensitive program. This restriction forces the user to be more specific in their search because one character variation can cause words or phrases to be analyzed or interpreted in many ways that can be used for different purposes.

```
harrison@harrison-Satellite-S50-A:~$ /usr/local/hadoop/bin/hadoop jar /usr/local
/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep ~/input
~/grep_example3 'Server listening[.]*'
```

Figure 12 – Command for grep_example3

We type in the command below in Figure 12 in order to obtain the instances of the aforementioned phrase. In Figure 13, we see the amount of times that “Server listening” popped up on the screen is eight. When the command was executed the first time, it showed up as two times because only log file existed. Since we are using four identical log files, it makes sense that there are eight in total.

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example3/*
8       Server listening
```

Figure 13 – Output showing all instances of “Server listening”

Now, we can get into more detailed examples. This example is focused on retrieving the number of times users have tried to connect to the system; specifically, we are looking for “Connection from” among the archives in ~/input.

```
harrison@harrison-Satellite-S50-A:~$ /usr/local/hadoop/bin/hadoop jar /usr/local
/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep ~/input
~/grep_example4 'Connection from[.]*'
```

Figure 14 – Command for grep_example4

Once the command in Figure 14 is run, the output should up with 328 instances of the words “Connection from,” as shown in Figure 15.

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example4/*
328     Connection from
```

Figure 15 – Output showing instances of “Connection from”

Our last example is probably the most interesting one. The objective of this demonstration of the MapReduce program is to show how many times someone has attempted to log onto our system or network and have failed to do so.

```
harrison@harrison-Satellite-S50-A:~$ /usr/local/hadoop/bin/hadoop jar /usr/local
/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep ~/input
~/grep_example5 'Failed password[.]*'
```

Figure 16 – Command for grep_example5

It is important to realize several things with the output in Figure 17. When there was only one file, the output originally came out as 101 instances of “Failed password” with a capital “F.” During the testing, the command was run as “failed password” with a small “f,” which gave zero results. Since we have four files in total, the command was run again and the output shown below was displayed. At the time of analysis, we concluded that if 101 instances were showing up for one file, then 404 instances would show up for four files. To verify that the program is functioning properly, we copied the data inside one file to a .docx file and used the search option of Microsoft Word to support our findings. With those results, we could prove that MapReduce on Linux is working fine.

```
harrison@harrison-Satellite-S50-A:~$ cat ~/grep_example5/*
404     Failed password
```

Figure 17 – Output showing instances of “Failed password”

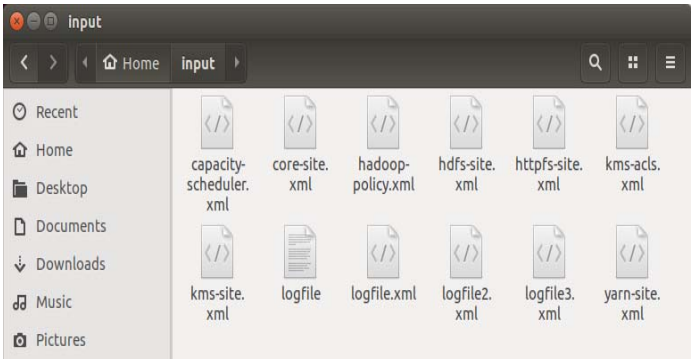

Figure 18 – ~/input directory with all files

Figure 18 shows the .xml files and log file as a .txt file that were used during experimentation. They are all stored in the ~/input directory.

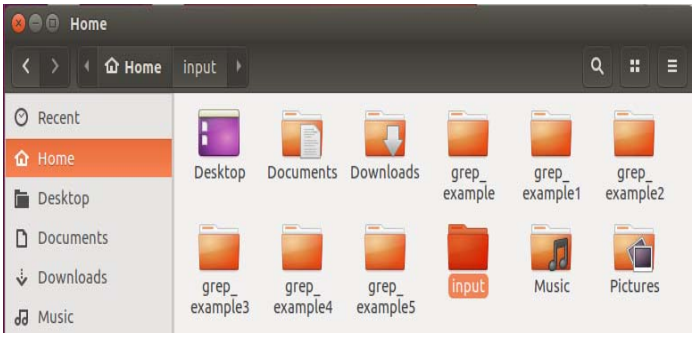

Figure 19 – home directory where all grep-example folders and input folder are stored

In Figure 19, you can see where all the grep_example folders are stored as well as the input folder where our .xml and .txt files are placed in. Each grep_example folder is created after each command is executed on the Linux command prompt. Once the grep command is put into use, a

text file is created inside each folder containing the instances of the stated word or phrase.

VI. CONCLUSION

During this investigation, we have uncovered a technique that could help us as manage and analyze any given data. Hadoop is one aspect that helps us to perform this task. The program used throughout this investigation was MapReduce, which helped us capture certain instances where a user may have tried accessing a network or an unauthorized person may have attempted an attack on the infrastructure by means of an activity log file. Although what was used in this experiment was that of a prototype, this small scale experimentation gives us the basic overview of what the application is capable of doing on greater scales for larger amounts of data stored on bigger systems. The use of Hadoop and MapReduce could help facilitate the management of large quantities of data and help us locate any form of information that may need further analysis or inspection.

REFERENCES

[1] Barot, G., Mehta, C., Patel, A.: Hadoop Backup and Recovery Solutions. O'Reilly, Sebastopol, CA (2015)
[2] Sitto, K., Presser, M.: Field Guide to Hadoop. O'Reilly, Sebastopol, CA (2015)
[3] White, T.: Hadoop: The Definitive Guide. Second Edition, Yahoo Press, (2009)
[4] Schneider, R.: Hadoop for Dummies. John Wiley & Sons, Inc. (2012)
[5] Frampton, M.: Mastering Apache Spark. Packt Publishing, Birmingham, UK (2015)
[6] Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark. O'Reilly, Sebastopol, CA (2015)
[7] Apache Hadoop http://hadoop.apache.org/ (Retrieved: 2017-03-17)
[8] Anderson, Melissa.: https://www.digitalocean.com/community/tutorials/-how-to-install-hadoop-in-stand-alone-mode-on-ubuntu-16-04/ (Published: 2016-10-13) (Retrieved: 2017-03-17)