# Overview of Hadoop and Use Hadoop on a Cluster

Lacey Conrad
MSDS 610
Regis University

May, 2020

## 1 Introduction

Hadoop has become a very popular programming framework to quickly process and store large data sets. Hadoop accomplishes this by distributing processing of the large data sets across clusters of computers known as nodes. The computing power of Hadoop is related to the number of nodes employed in a cluster; the more nodes used, the greater the computing power you have. Hadoop also is highly tolerant of hardware failure. If a node goes down, the data and jobs are redirected to other nodes since multiple copies of the data have been stored. Additional benefits of Hadoop include flexible data storage, free open-source framework, and easy scalability by adding additional nodes with increasing data size ("What is Hadoop?," 2020).

MapReduce is an element of Hadoop which prevents data ingestion bottlenecks by parallelizing the data analysis. Also, instead of bringing the data to where analysis occurs, the analysis is instead brought to the data. This is one of the most important aspects of MapReduce; data does not have to be moved since it can be analyzed on nodes. One common introductory use of MapReduce is to perform a word count on a text document. In the word count example, a program called a mapper creates key-value pairs out of the words in a text (initially, the values are all 1). The keys are then sorted and condensed into like keys, while incrementing the value each time a key is encountered (Lockwood, 2019).

This week, our focus was on implementing a MapReduce word count executed by Hadoop on a cluster of virtual machines. We began the lab by creating a cluster on Google Cloud Platform of 1 master and 2 worker nodes. SSH was used to open a terminal, where I was able to follow a similar procedure as I did in week 1 where I used a simple MapReduce program to perform a word count on a Shakespeare text. When I was confident the output from this week was nearly (if not completely) identical to last week, I began to investigate ways to clean the text of punctuation and stop words to enable the output of MapReduce to be more useful as an analytical tool.

## 2 Written Portion - Questions from week 2

1. What is a Hadoop distribution? What are some distributions of Hadoop?

   At its basic level a hadoop distribution includes two major components: 1) MapReduce, which is a data processor, and 2) data storage, known as Hadoop Distributed File System or HDFS. Hadoop is considered distributed since the Hadoop framework divides data among one or more nodes in a cluster. The data is then processed in parallel on the nodes where each data block resides (Schaefer, 2019). There are many different Hadoop distributions, with each having its own strengths and weaknesses related to the goal of the data analysis. Three commonly used distributions are the following:

   (a) Cloudera CDH: CDH was built for enterprises and includes Apache Hadoop, Apache Spark, Apache Kafka, and several other open source projects, all of which are integrated within Cloudera (Rosencrance, 2019).

   (b) Hortonworks Data Platform: HDP is an open source environment which is comprised of projects built by the Apache Software Foundation. Data is stored, processed, and analyzed in many formats and from many sources. It is also interesting to note that in 2019, Hortonworks merged

with Cloudera, and plans are underway to create a unified product called Cloudera Data Platform (Rosencrance, 2019).

(c) MapR: The MapR Data Platform enables simultaneous analytics and applications with real-time data analysis. The goal of MapR is for a user to be able to manage and entire data ecosystem from one platform.

2. How does a Hadoop cluster work?

   A computer cluster refers to a collection of computers that act together as a single unit. In the case of Hadoop, this concept extends to independent components connected by a network which work together as a unit. The Hadoop cluster acts as a computational cluster for data for storage and analysis, and does so as a distributed environment. Because the data analysis and workload is distributed across cluster nodes, the data can be processed in parallel ("Hadoop cluster overview: What it is and how to setup one?," 2017). The distribution of data across compute nodes is handled by the Hadoop Distributed File System (HDFS) when the `hadoop dfs -copyFromLocal` command is executed. Each time a file is copied into HDFS, the data is separated into chunks and replicated (for reliability) (Lockwood, 2019).

3. Why does Hadoop create multiple output files? How does this relate to the reducer step and number of compute nodes?

   Since data is stored in pieces on the compute nodes, parallel worker functions are sent to the nodes where the data pieces reside, and perform their analyses. As such, output files are created for each node, which will be recombined by Hadoop and HDFS. Specifically, this is accomplished via the Partitioner during the shuffle step of a MapReduce. The mapper takes in raw input, and transforms it into a series of key-value pairs, which are sent to the Partitioner to be assigned a reducer. The partitioner hashes the key, which will always be the same for a given key. This causes duplicate keys to end up on the same reducer. The reducer simply sorts the keys, and the values of a given key will be analyzed before moving to the next key (Lockwood, 2019).

4. Why should we use version control to store code for projects?

   Version control is a type of software that helps users manage code revisions and alterations. This allows modifications to the code to be tracked, allowing users to turn back the code to a previous point in time if a mistake is made, without disrupting other team members using the same code ("What is version control — Atlassian git tutorial," 2020). Version control software offers three primary benefits:

   (a) Change history of every data file. Since the changes made can be logged over the course of years, this allows users to use previous versions to identify root causes to bugs.

   (b) Branching and merging. This allows multiple streams of work to occur concurrently, while allowing for that work to be merged back together in such a way where it can be verified that changes do not conflict with each other.

   (c) Traceability. The ability to annotate each change with purpose and intent of the change allows for quick root cause analysis and bug tracking. Also, having the annotated changes allow users to understand previous changes, while producing changes that are in-line with the design of the system ("What is version control — Atlassian git tutorial," 2020).

5. What are pros/cons of using a cloud provider for a Hadoop cluster vs using our own in-house machines?

   **Cloud-based Hadoop - Positives to use:**

   (a) Cost: Using Hadoop in the cloud is cheap, and you only pay for what you use. A cloud provider will also take care of the infrastructure upkeep and security, requiring a company using the cloud provider to need less manpower to manage their Big Data.

   (b) Scalability: Hadoop in the cloud can be easily scaled up or down without incurring much more cost. This is not true with in-house Hadoop, where scaling up means purchasing more infrastructure.

   (c) Access anywhere: you can connect to Hadoop anywhere there is internet access (Shaik, 2017).

(d) Secure against disasters: Data can also be stored in the cloud, resulting in less data loss as the result of disasters, fire, hardware failure, etc (Eichkorn, 2019).

**Cloud-based Hadoop - Negatives to use:**

(a) Control: When using cloud-based Hadoop, you have no control of the systems hardware and very little control of the software (Shaik, 2017).

(b) Security and Privacy: Security is not as strong as in-house Hadoop (security exists at the software level for users), and third-party cloud services can have access to the data, indicating data privacy is also an issue (Eichkorn, 2019).

(c) Service outages: If your local internet or the cloud provider servers go down, you wont have access to your data. It is also possible to lose access to your data if the cloud provider is hacked, such as with a Denial-of-Service attack (Eichkorn, 2019).

**In-house Hadoop - Positives to use:**

(a) Security/Privacy: Since the data are located in-house, it is easier to monitor and make secure (Shaik, 2017). In addition, no third party can have access to your data (Eichkorn, 2019).

(b) Control: You have physical control of your infrastructure (Eichkorn, 2019).

(c) Elasticity: For non-elastic workloads, in-house infrastructure can often be cheaper (Shaik, 2017).

(d) Performance: For very small deployments, the performance of in-house Hadoop can actually be better than the performance on a cloud (Shaik, 2017).

**In-house Hadoop - Negatives to use:**

(a) Cost: Requires a large investment upfront to set up infrastructure (Eichkorn, 2019).

(b) Maintenance: In-house infrastructure for Big Data requires multiple servers, room to store them, and other necessities to keep it running, not to mention a host of people to ensure maintenance.

(c) Data back-up: In order to prevent complete loss of data, data needs to be backed up off-site or on the cloud (Eichkorn, 2019).

6. What do our top word counts look like now, and what do they tell us?

   Our counts now reflect words that are more meaningful for the text they derive from. If 5 completely different English texts were analyzed side-by-side, we would find that they would share many of the most frequent words. This is the point of removing stop words; to remove words that are so common they don't give us any context or meaning of a sentence. Similarly, punctuation doesn't provide any meaning or context for a sentence. Lastly, setting all the case to lower for all characters makes sure we're counting the keys accurately. After removing stop words and punctuation, the top words become thou, thy, shall, thee, good, lord, etc., and can actually provide context, sentiment, topic and meaning of the text (I have a table in the results section showing the word counts using the different types of filtering for reference). If this analysis were performed on customer feedback, for example, the most frequently used words could ascertain quickly if customers were generally having good experiences, or the words could be used to clue into a potential problem. For example, if Pizza Hut did a text analysis and found that a top word was rotten, they would want to get to the bottom of why that was the case quickly.

# 3 Methods and Code

## 3.1 Setting up a Google Cloud Platform cluster

In order to set up a Google Cloud Platform cluster, I began by logging into my Google cloud account. I then opened up a window to customize a cluster by navigating through the Dataproc sub-menu under the Big Data menu. Once in the create cluster window, I changed a few values from their default settings, and added the scikit-learn package by initialization as indicated in the lab handout. I then clicked create, and my cluster was created:

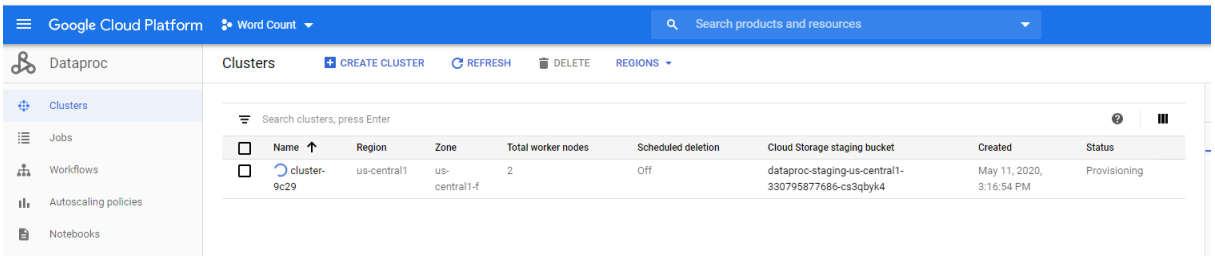


Figure 1: Creation of a cluster.

To open a terminal window, I clicked on the clusters name, selected the VM Instances header, and clicked on the SSH button on the right side of the master node:

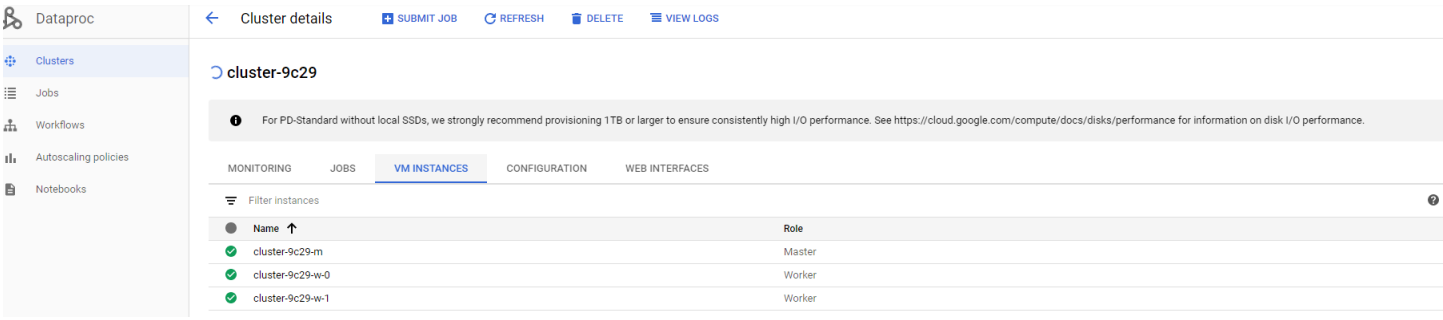


Figure 2: Google Cloud Platform cluster showing the master node and 2 worker nodes.

At this point, I was able to begin my text analysis using Hadoop.

## 3.2 Text analysis preparation

To begin the text analysis, I cloned into the courses github repository to download the correct code for the lab. I then installed the text document I will be performing analysis on, `shakespeare,txt`.

```
# Cloning the git hub repo where the MapReduce code is located:
$ git clone https://github.com/Regis-University-Data-Science/
simple_Hadoop_MapReduce_example.git

# Downloading the Shakespeare text:
$ wget http://norvig.com/ngrams/shakespeare.txt
```

Next, I created the file structure to house the input and output within HDFS using a series of `-mkdir` and `-copyFromLoval`:

```
# Making the directories for where the raw text will be saved:
$ hdfs dfs -mkdir /shakespeare
$ hdfs dfs -mkdir /shakespeare/input

# Making a copy of the text from the local file system into HDFS:
$ hdfs dfs -copyFromLocal shakespeare.txt /shakespeare/input

# Double check that our input is in the correct place:
$ hdfs dfs -ls /shakespeare/input
```

## 3.3 MapReduce streaming and viewing results

Since Python2 is installed by default on the Google Cloud Platform clusters, I had to run a slightly different version of the MapReduce streaming command than last week:

```
# Navigate into the directory for the Hadoop MapReduce example
# provided in class:
$ cd simple_Hadoop_MapReduce_example

# Streaming command to run the MapReduce on shakespeare.txt:
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /shakespeare/input \
-output /shakespeare/output
```

The streaming command and the commands to inspect the data (seen below) remain unchanged throughout the various MapReduce implementations used in this exercise.

```
# View the files in the output directory:
$ hdfs dfs -ls /shakespeare/output

# Moving up one directory
$ cd ..

# Merging the results from the reducers:
$ hdfs dfs -getmerge /shakespeare/output/ /home/lkconrad12580/result

# Viewing the results of my MapReduce by sorting from most
# counted key to least, and viewing the top ten of those:
$ sort -gr -k 2 result | head
```

**NOTE:** Before running the MapReduce streaming command and after I had saved or taken screen captures of the data, I ran the following code to clear the output directory, else MapReduce would not work.

```
# To remove the output directory:
$ hadoop fs -rm -r /shakespeare/output
```

### 3.4 Text cleaning - Implementation-1

Here is a initial implementation of a improved `mapper.py` that removes a few stop words and sets the cases of all characters to lower. While still not ideal, the output produced from this code will remove several overly used English words. In addition, all characters will be the same case, which will prevent cases such as "Soldier" being counted separately from "soldier".

```
#!/usr/bin/env python
import sys

# Creating a list of stop words:
stopwords = set(['the', 'and'])

# Reading in all lines from stdin:
for line in sys.stdin:

        # Removing leading and trailing whitespace:
        line = line.strip()

        # Setting all characters to lowercase:
        line = line.lower()

        # Splitting each line into words; splitting on any whitespace:
        words = line.split()

        # Creating output tuples (word, 1) in tab-delimited format,
```

```
        # appropriate format for \code{reducer.py}:
        for word in words:

                # Only print the tuple if the word is not listed
                # in stop words:
                if word not in stopwords:
                        print '%s\t%s' % (word, "1")
```

### 3.5 Text cleaning - Implementation-2

To further refine the `mapper.py` program, I utilized the stop word list from `scikit-learn`, and added code to remove punctuation. Since this step removes punctuation, it will no longer take up processing time or take up space in our word count. Additionally, this will also prevent cases like "solider," being counted differently than "soldier". The additional stop words in the sklearn list will be skipped over when creating key-value pairs, resulting in an analysis that is not bloated with meaningless keys.

```
#!/usr/bin/env python
import sys
from sklearn.feature_extraction import stop_words
import re

# Creating a list of stop words from sklearn:
stopwords = set(stop_words.ENGLISH_STOP_WORDS)

# Reading in all lines from stdin:
for line in sys.stdin:

        # Removing leading and trailing whitespace:
        line = line.strip()

        # Removing punctuation using regex:
        line = re.sub(r'[^\w\s]', '', line)

        # Setting all characters to lowercase:
        line = line.lower()

        # Splitting each line into words; splitting on any whitespace:
        words = line.split()

        # Creating output tuples (word, 1) in tab-delimited format,
        # appropriate format for \code{reducer.py}:
        for word in words:

                # Only print the tuple if the word is not listed
                # in stop words:
                if word not in stopwords:
                        print '%s\t%s' % (word, "1")
```

## 4 Results and Output

In the following table, I have displayed the top ten words in the original `mapper.py` program (SIMPLE), followed by some stop word removal and text case normalization (IMPL-1), and lastly, removal of stop words using sklearn, in addition to removal of punctuation (IMPL-2):

SIMPLE and IMPL-1 look similar as both have 3/10 words actually as punctuation, and the remaining words are just shuffled some between the two due to text case normalizing. For example, "i" moved up since all "i's" were included in the same key. This was the same theme for the other words on the IMPL-1 list.

| | SIMPLE | | IMPL-1 | | IMPL-2 | |
|---|---|---|---|---|---|---|
| rank | key | value | key | value | key | value |
| 1 | , | 81827 | , | 81827 | thou | 5357 |
| 2 | . | 36514 | . | 36514 | thy | 3811 |
| 3 | the | 23272 | i | 20042 | shall | 3603 |
| 4 | I | 20041 | to | 18533 | thee | 3103 |
| 5 | ; | 17274 | ; | 17274 | good | 2817 |
| 6 | and | 16817 | of | 16007 | lord | 2717 |
| 7 | to | 15506 | you | 13834 | o | 2615 |
| 8 | of | 15037 | a | 13679 | come | 2565 |
| 9 | you | 12361 | my | 12257 | sir | 2541 |
| 10 | a | 12155 | that | 10719 | let | 2108 |

Table 1: The top ten words (keys) for each implementation of MapReduce on `shakespeare.txt`. SIMPLE refers the the original `mapper.py` code included with the lab, IMPL-1 (implementation-1) is an initial attempt at text cleaning within the `mapper.py` program, and IMPL-2 (implementation-2) is text cleaning including setting text to lower case, punctuation removal, and stop word removal.

When comparing SIMPLE, IMPL-1 and IMPL-2, IMPL-2 shares no words with the other implementations. The presence of "o" in IMPL-2 is an oddity, and likely related to the manner in which Shakespeare wrote. It wouldn't be difficult to add it as a stop word, but for reasons discussed below, I decided to leave it in the word count.

# 5 Analysis

Proper application of cleaning techniques to files before text-based analysis greatly clarifies the context, meaning and sentiment of a body of text. While there are many ways to clean a text, we focused on removing stop words and punctuation in the current exercise. Stop words and punctuation will take up storage space in databases and will increase processing time during analysis even though we generally receive no useful information from them in our analysis ("Removing stop words with NLTK in Python," 2017).

However, the issue of how to properly pre-process text for analysis is not as cut and dry as I make it out to seem above. It is easy to assume that 1) text cleaning is something we always do and 2) text cleaning should always been done as thoroughly as possible. Yet, one cannot forget that on occasion, stop words (and maybe even punctuation?) may in fact be useful depending on the type of analysis you are running and the type of text you are analyzing. Similarly, words can be empty of meaning in a text and not be considered a stop word. This is why there is no universal stop word list (Vallantin, 2019).

In the case of my word count, I feel that it is appropriate to say I present three options on the pre-processing of text instead of saying how well my second implementation cleaned data and produced meaningful words. As a method to produce potentially meaningful text in light of the exercise, implementation two clearly produces less "useless" words and symbols. Yet we are not performing any analysis besides a word count. I am not performing a sentiment analysis on customer data or trying to determine if Lady Gaga's latest album was well received. Therefore, it appears to be important to know how pre-processing will change the outcome of a text analysis in light of the subject being analyzed.

# 6 Conclusion

As is the case with any type of analysis, the quality of your results can be directly attributed to the quality of your data. Stop words are considered useless data, yet as I mentioned in my analysis, not all stop words, and not all texts, can be handled in identical manners. Text pre-processing, as is the case with data pre-processing, needs to be performed with knowledge on how it will change the results and with what the goal of the analysis is.

I found this topic to be very fascinating, and somewhat addictive. Maybe it's human nature, but I felt like there should be this clean-cut, perfect way to pre-process text that is universal, and that clearly isn't the case. Which I'm finding is a very good thing. If every text was analyzed the same way, then we are likely to get similar and possibly useless results, since we would not be refining the analysis based on what the text was, and what we were trying to learn from the text.

Moving past any of the concerns about how to (or even if you should) pre-process data, I found a lot of the processing methods very interesting. Not surprisingly, what we did in this lab are common methods in text pre-processing. It is still easy to find places that can be improved upon, such as the mysterious "o" in my results. This likely came as a result of punctuation removal from words that had a prefix of "O," which I believe Shakepeare enjoyed using. My research into text pre-processing showed that there are more accurate ways to normalize text for pre-processing. I kept finding references to stemming and lemmatization, which seem to be two commonly used methods. These methods seek to reduce inflectional and derivationally related forms of words to their base form. An example given in my reading is "am," "are," "is," would be reduce to "be." Or, "car," "cars," car's," and "cars"' being reduced to car. This seems like it could greatly enhance the quality of text analysis, since in the case of most types of text-based analysis, these different forms of car all really mean the same thing and shouldn't be counted differently.

# 7 References


1. Eichkorn, D. (2019, September 25). *How to identify the pros and cons of on-premise vs. cloud computing.* Gordon Flesch Company. `https://www.gflesch.com/blog/pros-and-cons-of-cloud-storage`

2. *Hadoop cluster overview: What it is and how to setup one?* (2017, July 1). DeZyre. `https://www.dezyre.com/article/hadoop-cluster-overview-what-it-is-and-how-to-setup-one/356`

3. Lockwood, G. (2019, October 20). *Conceptual overview of map-reduce and Hadoop.* Glenn K. Lockwood. `https://www.glennklockwood.com/data-intensive/hadoop/overview.html`

4. *Removing stop words with NLTK in Python.* (2017, May 23). GeeksforGeeks. `https://www.geeksforgeeks.org/removing-stop-words-nltk-python/`

5. Rosencrance, L. (2019, May 22). *The main picks for Hadoop distributions on the market.* SearchDataManagement.

   `https://searchdatamanagement.techtarget.com/feature/The-main-picks-for-Hadoop-distributions-on-the-market`

6. Schaefer, P. (2019, January 11). *Hadoop distribution — Future of Hadoop distributions.* Trifacta. `https://www.trifacta.com/blog/hadoop-distribution/`

7. Shaik, S. (2017, January 2). *Hadoop on cloud vs Hadoop on premises.* LinkedIn. `https://www.linkedin.com/pulse/hadoop-cloud-vs-premises-shahebaz-shaik/`

8. *Stemming and lemmatization.* (2009, April 7). The Stanford Natural Language Processing Group. `https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`

9. Vallantin, L. (2019, October 7). *Why is removing stop words not always a good idea.* Medium. `https://medium.com/@limavallantin/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214`

10. *What is Hadoop?* (2020). SAS. `https://www.sas.com/en_us/insights/big-data/hadoop.html`

11. *What is version control — Atlassian git tutorial.* (2020). Atlassian. `https://www.atlassian.com/git/tutorials/what-is-version-control`