Massive Data Processing - Lab 1

Emilie Leblanc, emilie.leblanc@student.ecp.fr

February 21, 2017

1 Hadoop setup

For my Hadoop setup, I have used a VirtualBox in which I have installed Ubuntu 16.04 and Java. I have also created a new user, **hduser**, on which I installed Hadoop 2.7.3.

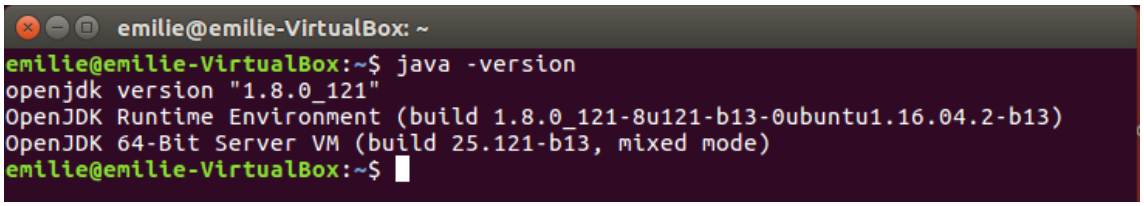


Figure 1: Version of Java installed.

In order to set up Hadoop, I followed various tutorials, including (but not restricted to):

- https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-in-stand-alone-mode-on-ubuntu-16-04
- https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
- https://www.youtube.com/watch?v=ve841JxF_VE

In these tutorials, I understood how to setup the different Hadoop files. In **hadoop-env.sh**, I set:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/
```

In **etc/hadoop/core-site.xml**, I set:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

In **etc/hadoop/hdfs-site.xml**, I set:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

I also create the **etc/hadoop/mapred-site.xml** file as:

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

And configured **etc/hadoop/yarn-site.xml** to:

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

In the hduser, after having formatted the filesystem with the **bin/hdfs namenode -format** command line, I started the **start-dfs.sh** and the **start-yarn.sh** files as seen bellow.

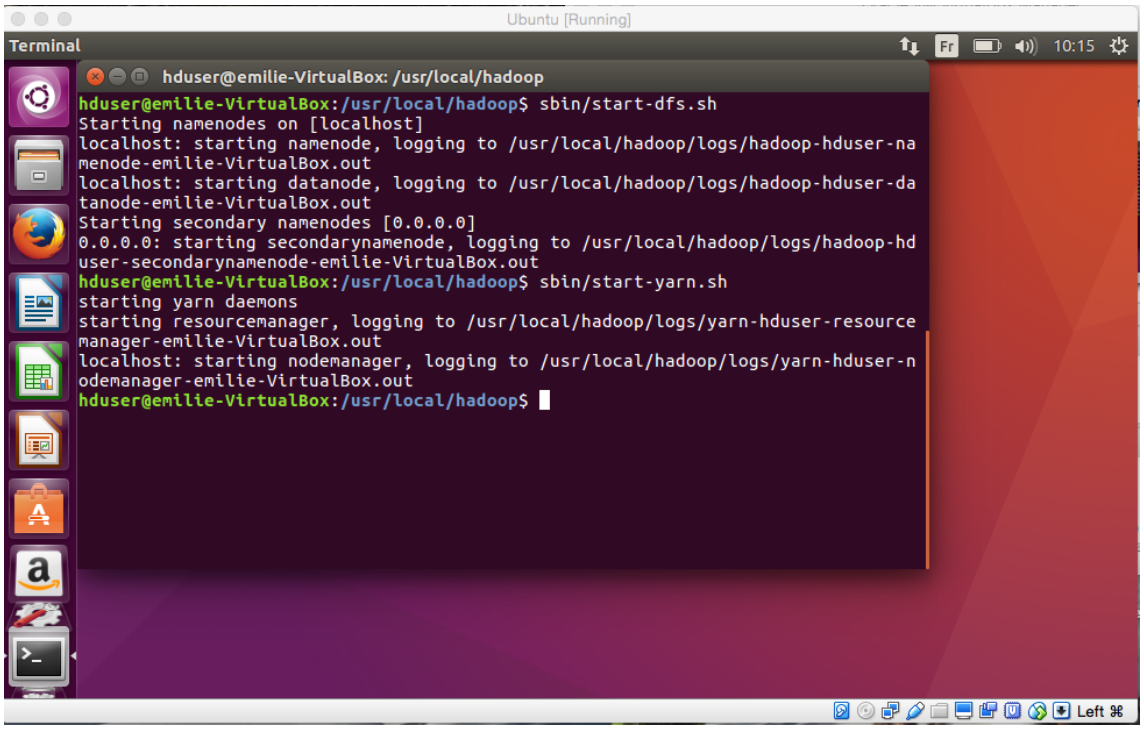


Figure 2: Starting processes.

And then verified that everything was running correctly with the **jps** command:

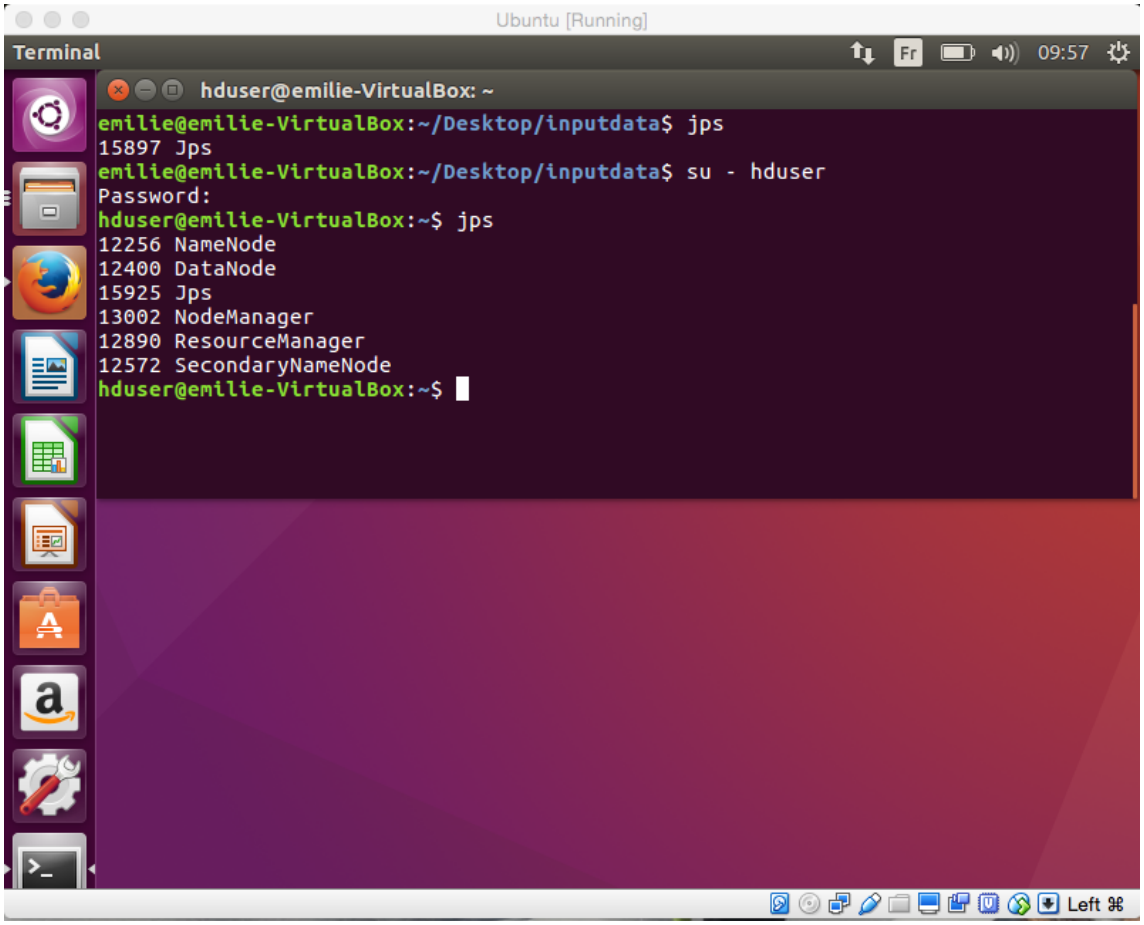


Figure 3: Jps command.

After having configured Hadoop correctly, I downloaded the datasets.

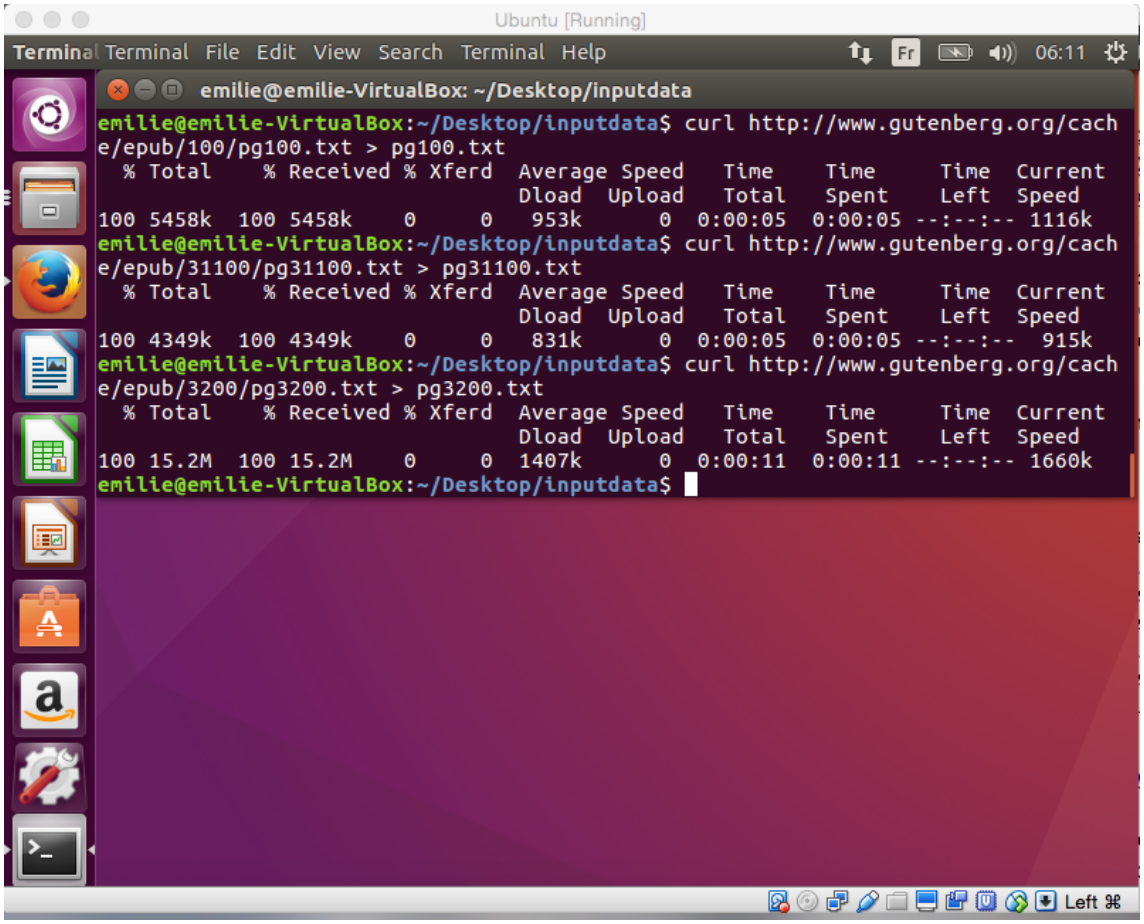


Figure 4: Downloading datasets from gutenberg.

2 Run a MapReduce to identify stopwords

In order to run a MapReduce to identigy stopwords, I created a **all.txt** file, containing all the lines of the **pg100.txt**, **pg31100.txt** and **pg3200.txt** files through the following command line:

```
1 cat input-files/* > all.txt
```

Then, I created a user directory in the HDFS and inserted the data.

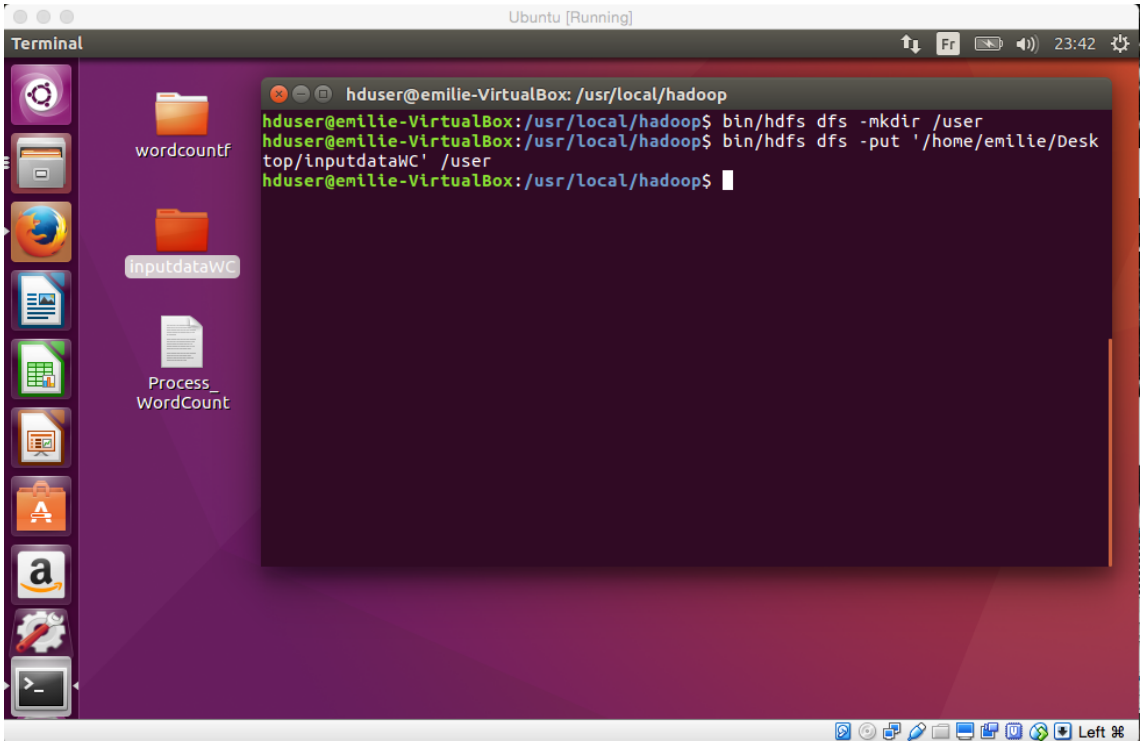


Figure 5: Creating directory in HDFS and inserting data.

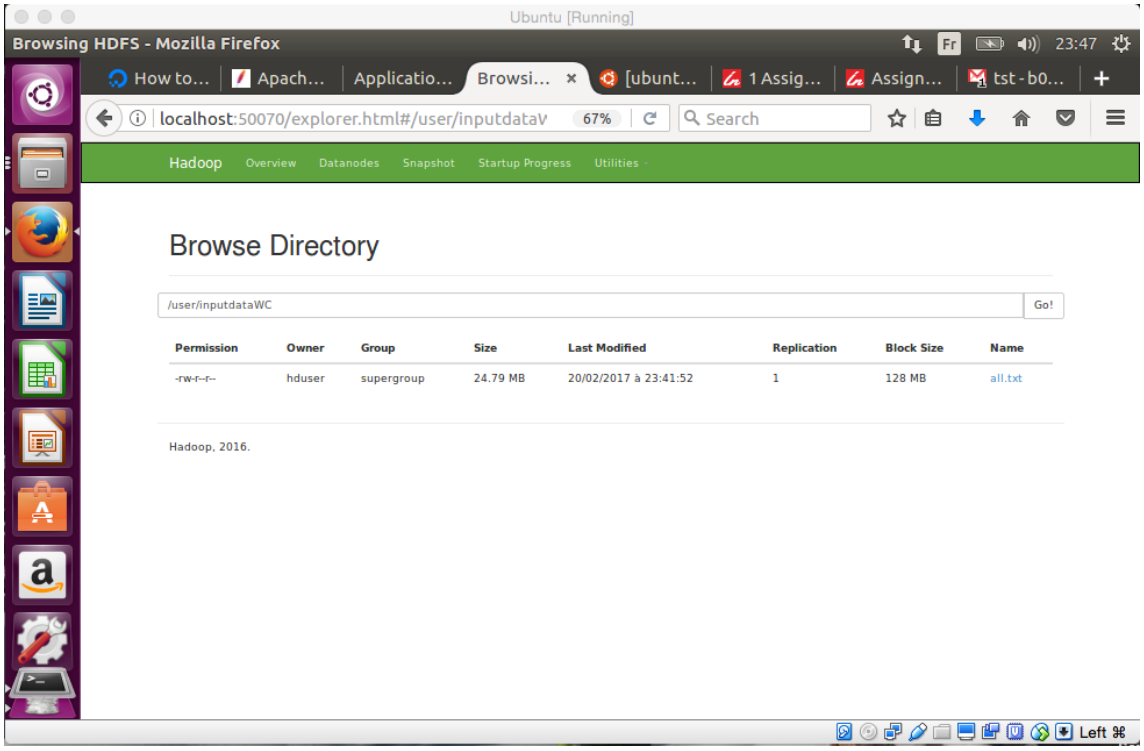


Figure 6: Checking that the data has been imported.

In the **wordcountf** folder on my Desktop, I inserted my MapReduce code for stopwords: **WordCountpro.java**. Placing myself in the path of the folder, I ran the following command (also found in the **Process_WordCount.txt** file). This allows us to create the different files necessary to create the jar, that we then place in a **wordcountc** file.

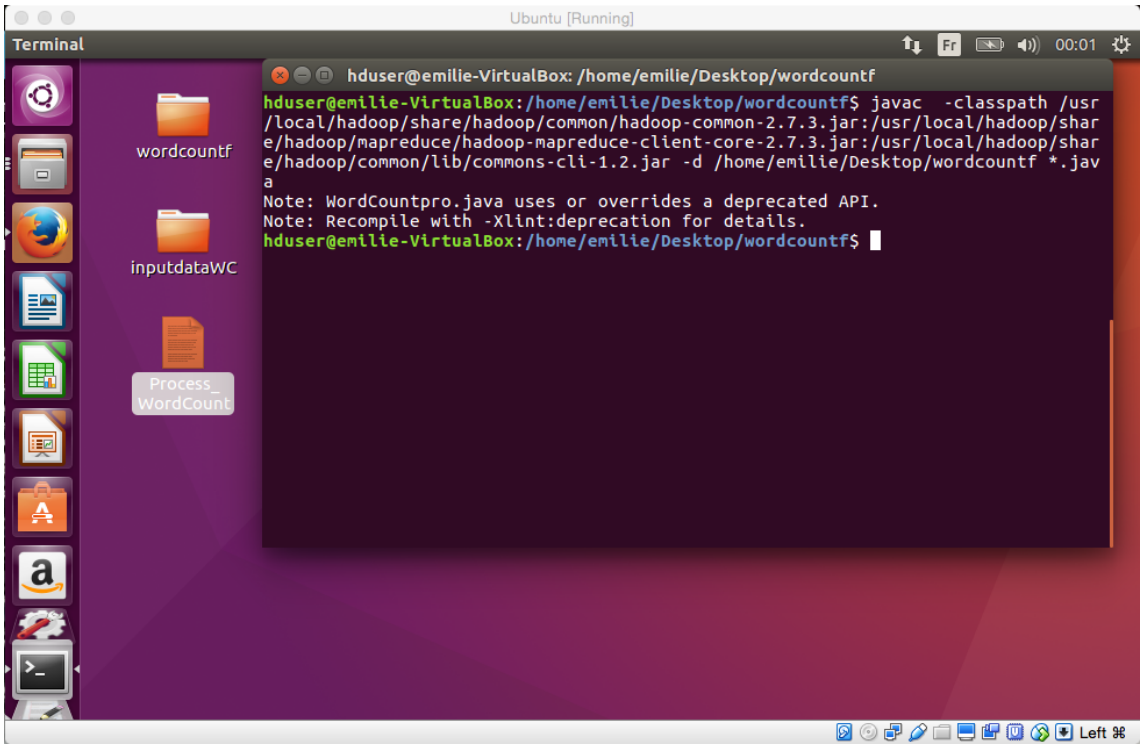


Figure 7: Creating intermediary files.

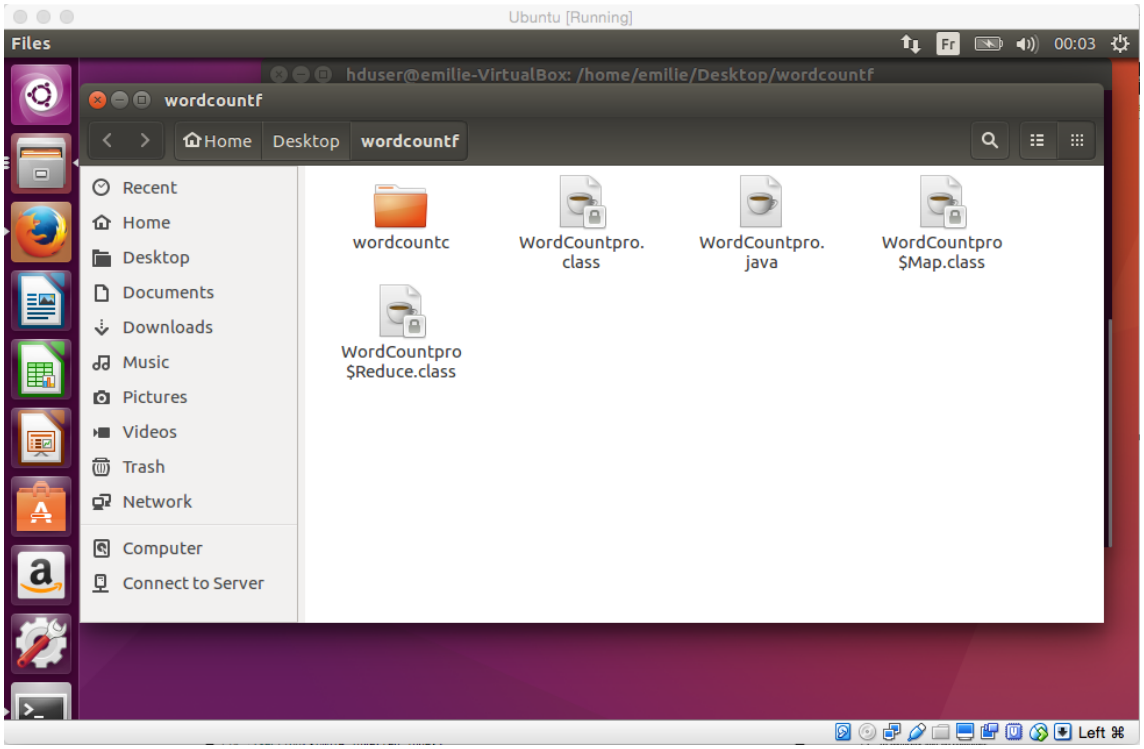


Figure 8: Placing the files in wordcountc.

Then, after having done the compiling, we convert the **wordcountc** folder in jar. As we can see, the operation succeeded.

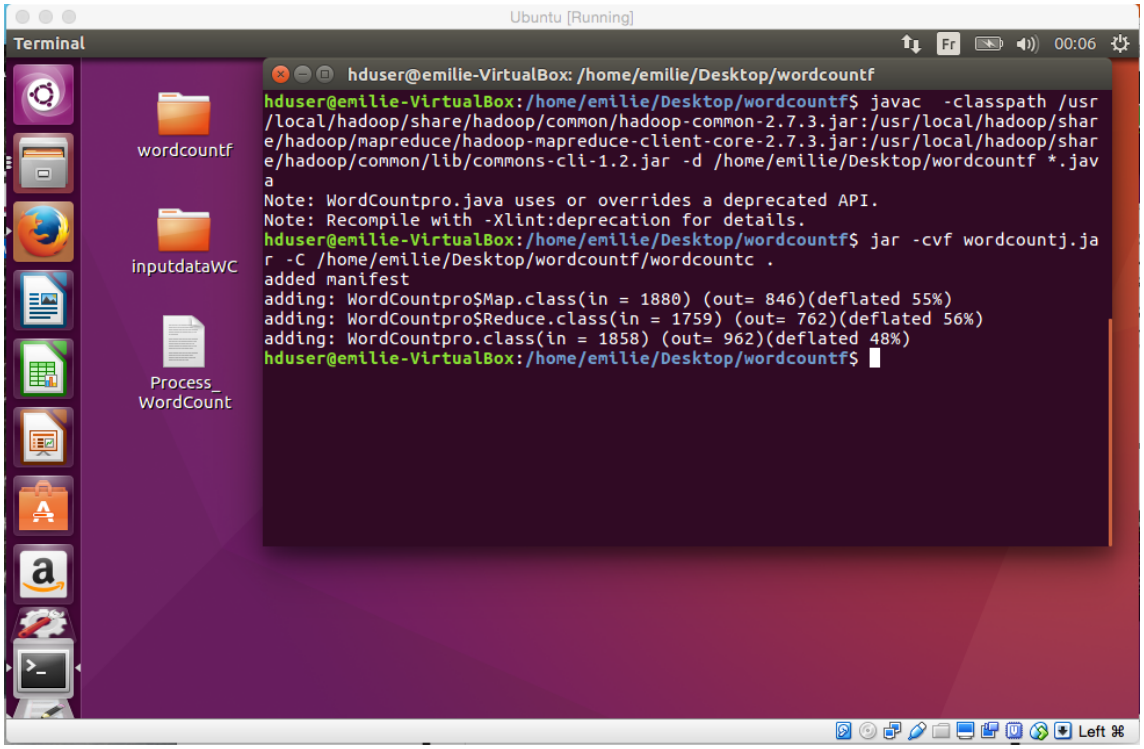


Figure 9: Converting the wordcountc folder to jar.

Now, we will be able to run the stopword MapReduce program (here called WordCountpro) whilst changing various settings.

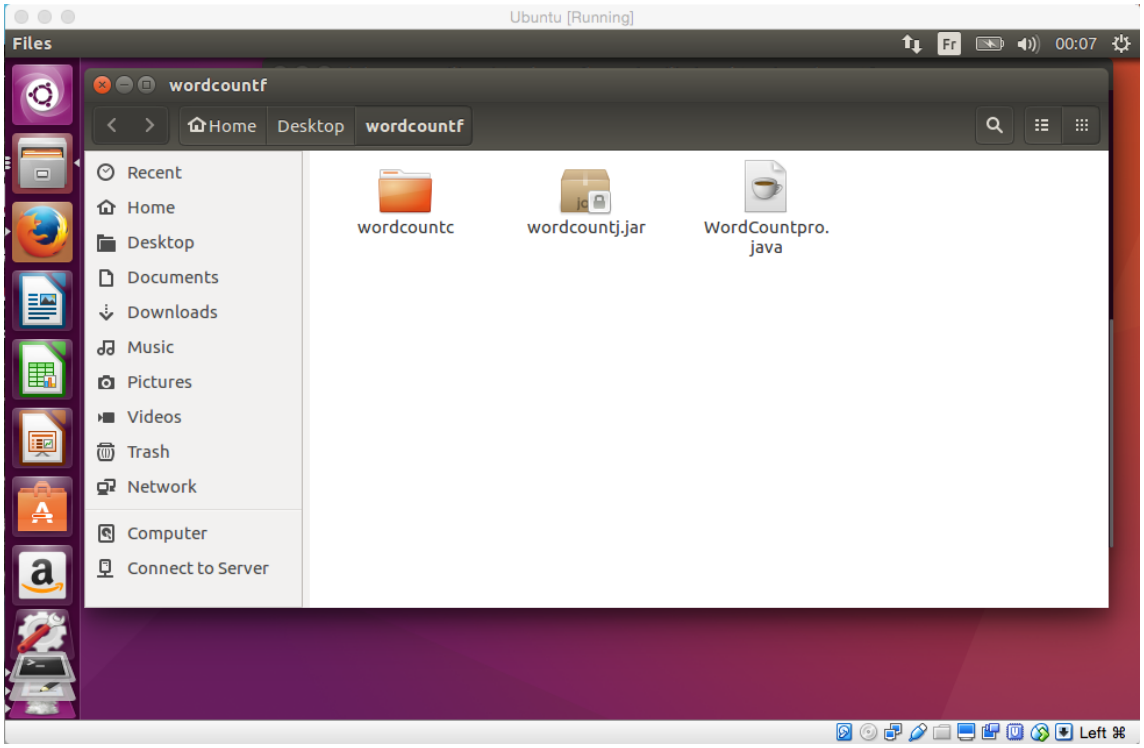


Figure 10: Checking jar creation.

2.1 10 reducers and no combiner

In order to run the MapReduce with 10 reducers and no combiner, we run the following command line in **/usr/local/hadoop**:

```
1 bin/hadoop jar /home/emilie/Desktop/wordcountf/wordcountj.jar
      WordCountpro /user/inputdataWC outputwc 10 0 0
```

See the **WordCountpro.java** and the **Process_WordCount.txt** for full detail. As seen at localhost:50070, the job was a success.

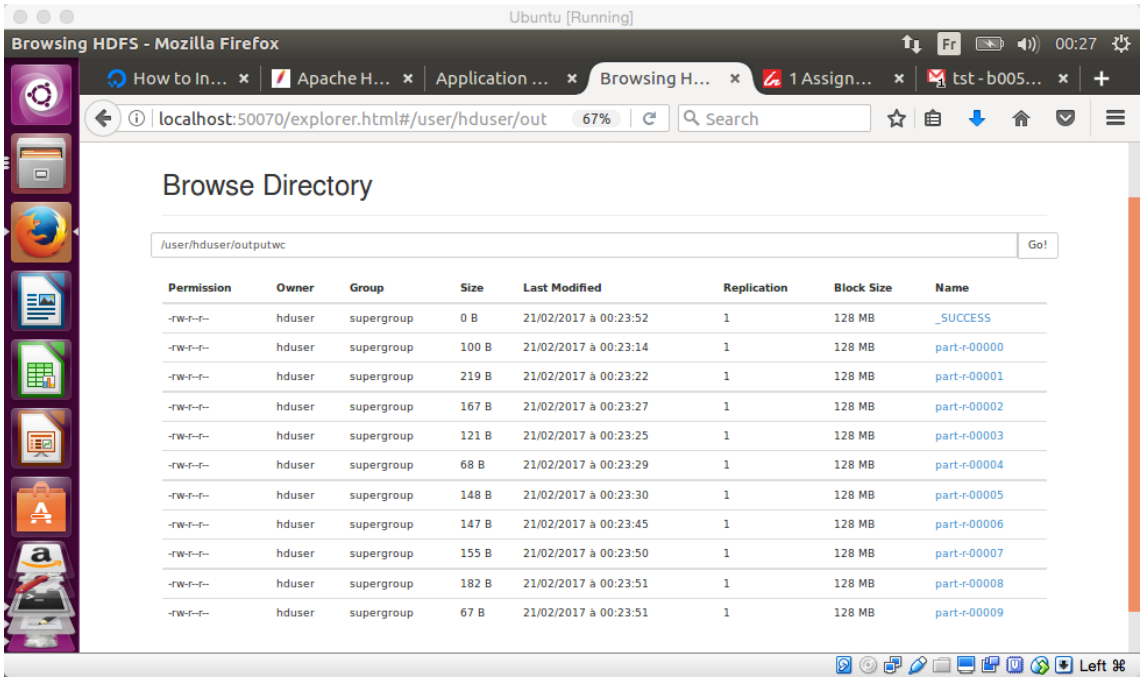
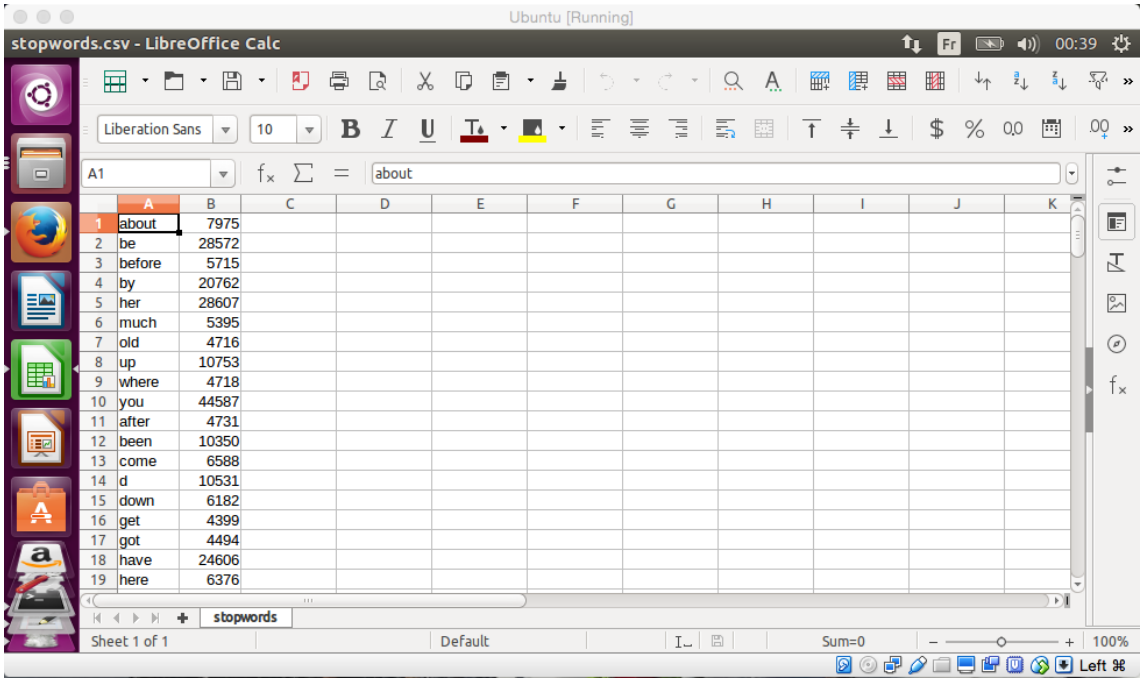


Figure 11: Stopword job with 10 0 0.

And in order to create a full **stopwords.csv**file, we ran the following command lines to merge the files and transform the output in csv:

```
cat Downloads/* > Desktop/stopwords.txt
sed 's/\t/,/g' Desktop/stopwords.txt > Desktop/stopwords.csv
```



	A	B
1	about	7975
2	be	28572
3	before	5715
4	by	20762
5	her	28607
6	much	5395
7	old	4716
8	up	10753
9	where	4718
10	you	44587
11	after	4731
12	been	10350
13	come	6588
14	d	10531
15	down	6182
16	get	4399
17	got	4494
18	have	24606
19	here	6376

Figure 12: Extract of the stopwords.csv file.

As we can see at **localhost:8088**, the job took **1min37sec**.

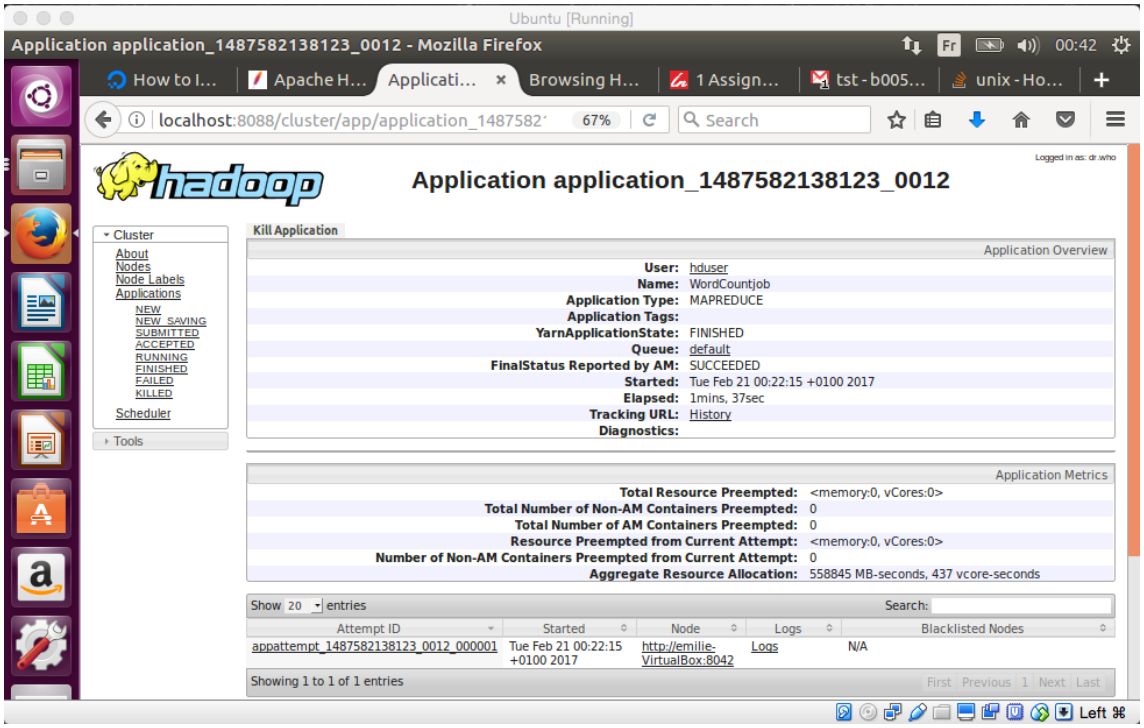


Figure 13: 10 0 0 time.

2.2 10 reducers and 1 combiner

In order to run the same process with 10 reducers and 1 combiner, we run the following command line, after having deleted the previous output files from the hdfs:

```
1 bin/hadoop jar /home/emilie/Desktop/wordcountf/wordcountj.jar
    WordCountpro /user/inputdataWC outputwc 10 1 0
```

As we can see, this time the job took **1min14sec**.

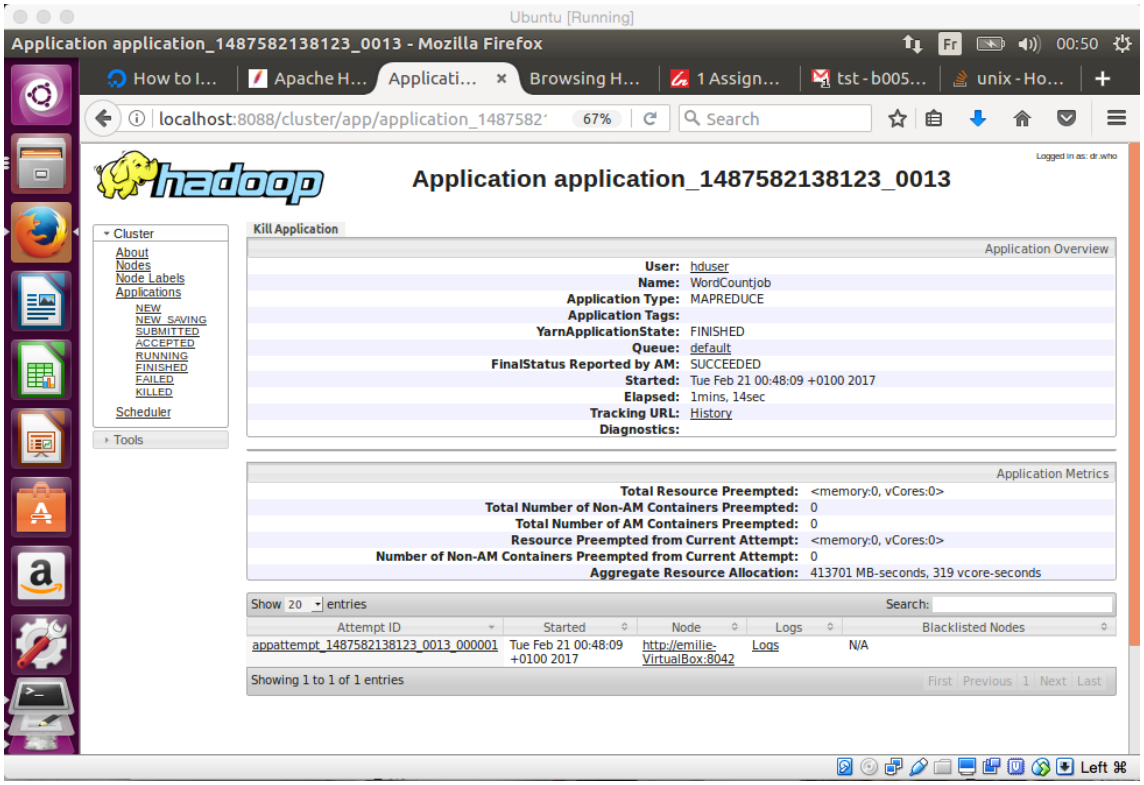


Figure 14: 10 1 0 time.

2.3 10 reducers and 1 combiner with compression

We proceed the same way with:

```
1 bin/hadoop jar /home/emilie/Desktop/wordcountf/wordcountj.jar
    WordCountpro /user/inputdataWC outputwc 10 1 1
```

And the output is of **1min17sec**.

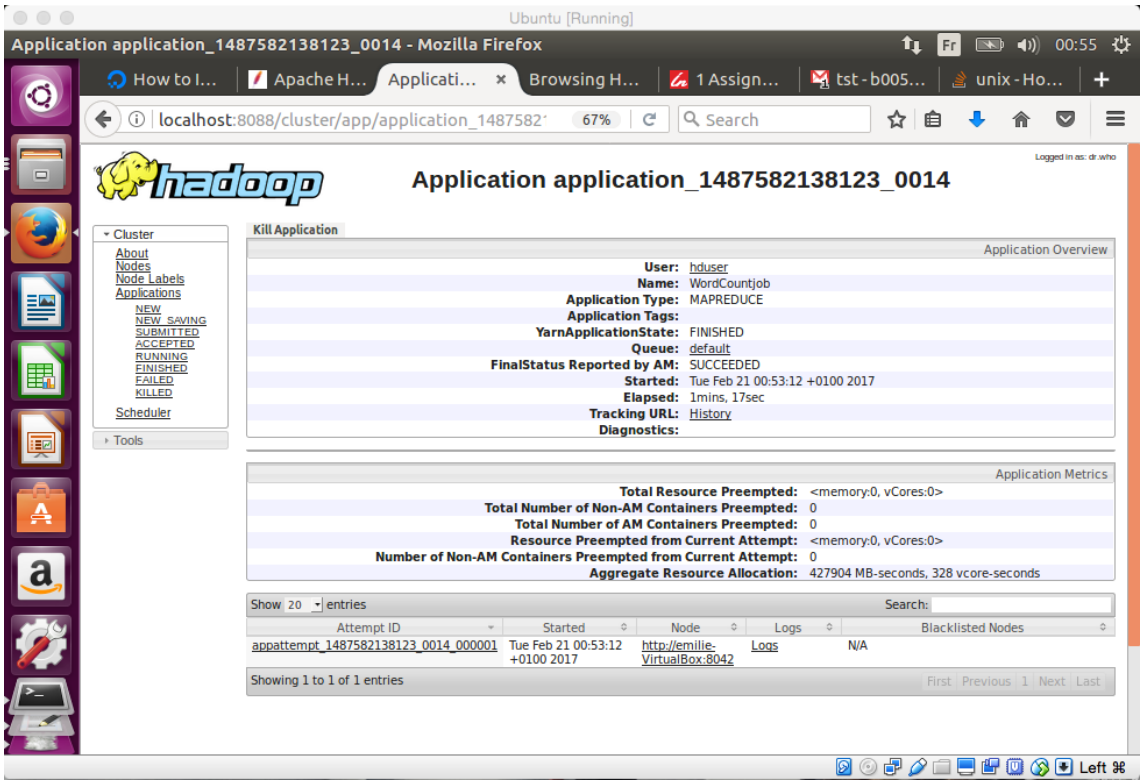


Figure 15: 10 1 1 time.

2.4 50 reducers and 1 combiner with compression

Finally, with 50 reducers, a combiner and compression, we ran:

```
1 bin/hadoop jar /home/emilie/Desktop/wordcountf/wordcountj.jar
    WordCountpro /user/inputdataWC outputwc 50 1 1
```

And the final output was of **4min17sec**.

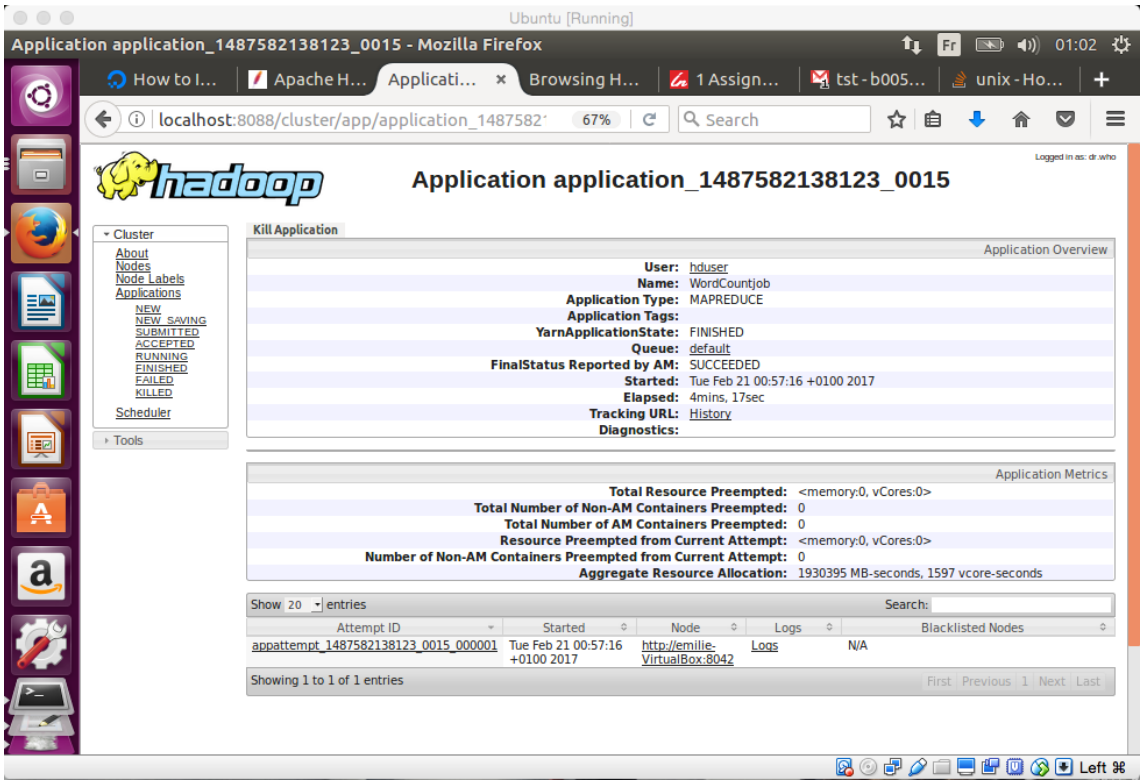


Figure 16: 50 1 1 time.

To conclude:

- Adding a combiner reduces the computation time. This is because the combiner reduces the output pairs of each mapper and thus lowers the load for the reducers.
- Adding compression increases the computation time compared to simply having 10 reducers and a combiner. Compression mainly reduces network transfer times, so it is not useful on a single node cluster.
- Going up to 50 reducers largely increases the computation time. Since we are splitting the output in too many ways, each mapper has to save its output for all the forthcoming reducers and thus reduces performance.

3 Simple and complex inverted index

We proceeded as before and managed to obtain the following results.

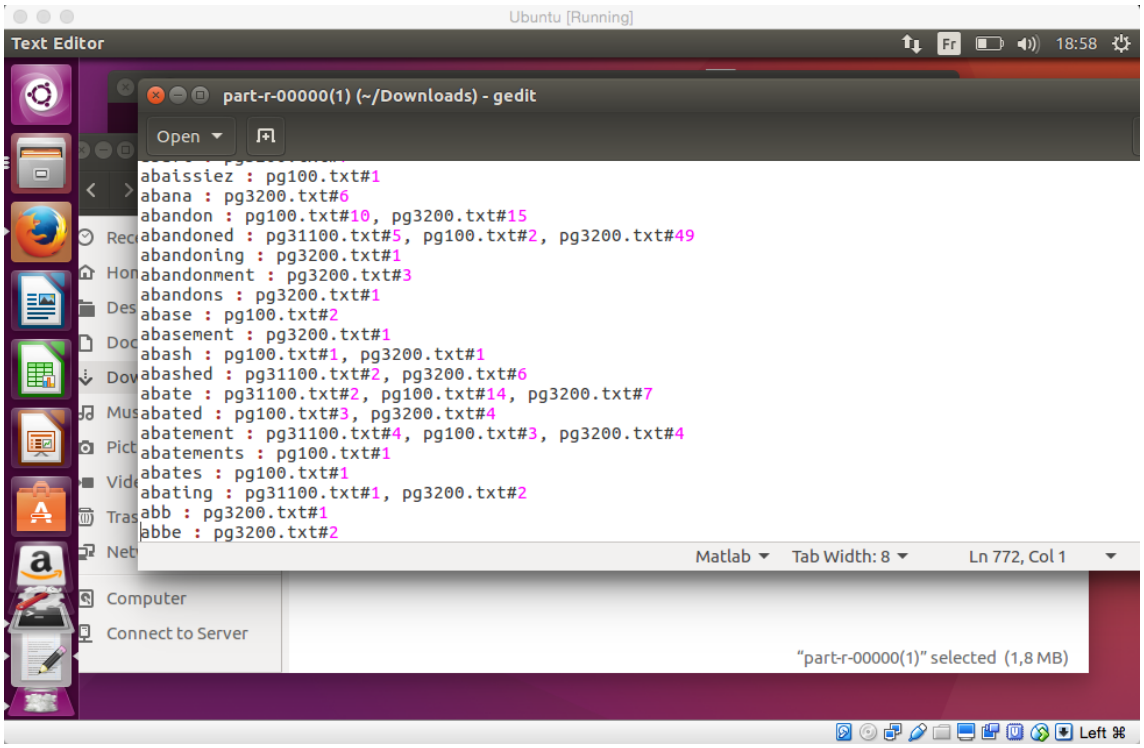


Figure 17: Good results obtained.

However, a bug is now occurring and, after numerous attempts to correct it, we are stuck with the following error:

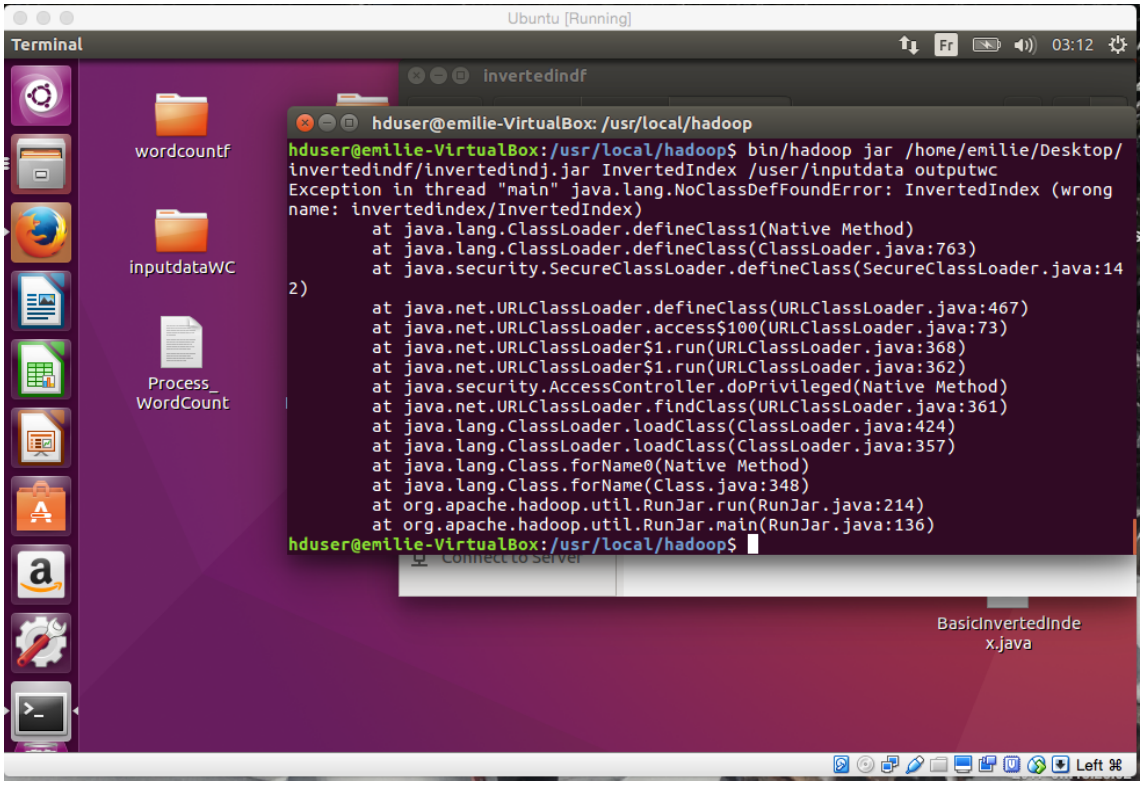


Figure 18: Error currently obtained.

4 Number of unique words

In order to do so, we have to create a counter such as:

```
1 public static enum CUSTOM_COUNTER {
2         UNIQUE_WORDS,
3       };
```

But we did not manage to make it work.