**Parker Hague – Assignment 1 Report**

**Part 1 SCREENSHOTS:**

**i: files and directories where the tweet data is stored in HDFS**

Text

Description automatically generated

Figure 1

This picture, **Figure 1**, shows the location of my downloaded Twitter data. As you can see, the data is located within several subfolders of the /user/phague/assn1/2022/03/04 directory.

I also wrote some java code that will list all the sub directories in my /user/phague/assn1 folder. The following, **Figure 2**, shows some of the output.

A screen shot of a computer

Description automatically generated with medium confidence

Figure 2

**ii: contents of file in HDFS that stores the tweet data**

The following image, **Figure 3**, shows the contents of one of the files containing twitter data. This data is displayed through an **hdfs dfs -cat**  command.

Text

Description automatically generated

Figure 3

**Part 1 APPROACH:**

To download the Twitter data I had to do several things. I had to first make a developer account with Twitter. I had to then create a new project so that I could use their access keys and tokens for me to be able to use their api. I had to then create an HDFS directory for my data to go in.

Next, I created a .conf file that had all the configuration for the Flume Agent to interact with the Twitter api. I chose my keywords to be “crypto” and “OSRS”. Finally, all the setup is done and I can download the data. I used an nohup command and started the data retrieval process. I let the data collection occur for about 10-20 minutes. I was then able to list the contents of my directory to see that I had downloaded the data successfully. I had then collected about 50MB of data.

Lastly, I wrote a couple of small functions in Java to print out my HDFS directory and print out the contents of an input file. I did this to ensure that my program was successfully reading the data. I included screenshots of me listing the files in HDFS and also listing the files from the Java program. Now that I have my data, I am ready to select some files for input for part2.

**Part 2 SCREENSHOTS:**

**Count rows using MapReduce**

**Figure 4**  shows a printout of the input files I am using for my MapReduce task as well as the accumulative file size. As you can see, my input is approx. 2.27MB.

Text

Description automatically generated

Figure 4

To count my number of tweets, I counted them by their occurrence of the string “created\_at” since this was specific to each tweet. I printed out the times created as well as the count for that specific tweet. **Figure 6** shows that there are a total of 705 tweets in my dataset.

Text

Description automatically generated

Figure 5

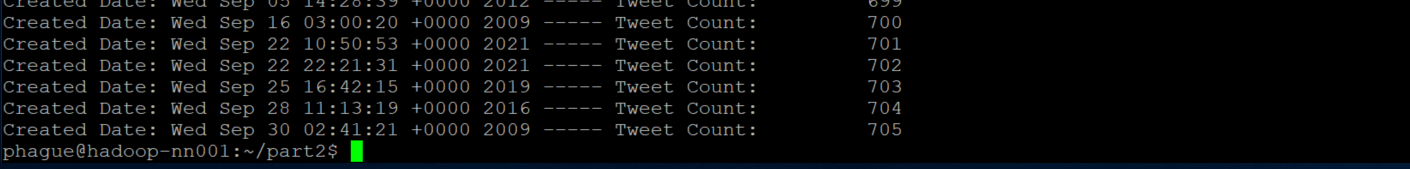


Figure 6

**Part 2 APPROACH:**

For this part we simply needed to count the number of tweets using MapReduce. The most challenging part of this was being able to parse out the JSON to see the separated Tweets. The JSON response for this api is extremely ugly and very nested so it took me a little while to parse through all the data. I ended up separating the Tweets by the “created\_at” property since every single tweet had that property.

Once I had finally separated the tweets, I was finally able to write a key value pair inside my map function. I used the created date as the key and just passed in 1 as the value. Here I just used 1 as a value because it will allow me to easily count the number of tweets by simply doing count++;

I then used my reducer to look at all of the keys. If a key appeared more than once, I just ignored it. This way I was able to count the number of distinct keys. I used a counter variable as my value for the reduce function. I then wrote the key value pair as the creation date and the count of the tweet respectively. This is how I was able to obtain the count of my tweets using MapReduce.

**Part 3 SCREENSHOTS:**

**HashTag Partitions**

A computer screen capture

Description automatically generated with low confidence

Figure 7

I partitioned my hashtags across many different reducers. **Figure 7** shows some of the output directory and **Figure 8** and **Figure 9** shows what the output of some of the files looks like. As you can see, the work was split up across multiple partitions. In these examples, each hashtag appeared in two tweets that contained the word “crypto”. I have also included a snapshot of my output without any partitioning, see **Figure 11**.

I also included **Figure 10** which is a list of each unique hashtag from the hashCount folder.

Text

Description automatically generated

Figure 8

Graphical user interface, text

Description automatically generated

Figure 9

Text

Description automatically generated

Figure 10

Text

Description automatically generated

Figure

**Part 3 APPROACH:**

This part took the longest in comparison to the other two. For this part we have to collect all of the hashtags and then count the occurrence of our keyword in tweets based on each hashtag.

In my map function, I first passed out the text in each tweet. I then also parsed out the hashtags in each tweet. I then collected each unique hashtag and queried it against each tweet to see which keyword the tweet contained. Once I knew each hashtag and which keyword corresponded to it, I was able to write this to a key value pair. I used the hashtag as the key and I used a counter 1 as the value.

The partitioner then takes over the job from the mapper. I looked at each distinct key and I decided which partition the key will belong to based on a round robin sequential style. I did this to ensure that the distribution is as even as possible. I also made sure that each partition had no more than 10 hashtags in it. Now that the function has decided which partition each key-value will have, we can now go onto the reducer.

Each partition will have its own reducer. My reducer takes in each key and sums up the occurrence of each value for that key. It then writes the reduced output to a file where the key is the hashtag and the value is the number of times it occurred with that specific keyword. In my case, there weren’t any tweets with the keyword “OSRS” so all of the occurrence values are 0 for that keyword.