DS 256 - Scalable systems in Data Science
Assignment - 2

Sheshadri K. R.
Dream Lab (CDS)
06-02-02-10-12-18-1-16336

April 16, 2019

1 Experimental Setup

System Environment of Turing Node:

- **Java Version :** 1.8.0_112 (Oracle Corporation)
- **Spark Version :** 2.1.1.2.6.1.0-129
- **Scala Version :** 2.11.8
- **CPU:** Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- **OS architecture :** amd64
- **Number of Processing Cores :** 12
- **Total installed RAM :** 47GB

HDFS Environment:

- **Block size :** 128MB
- **Replication Factor :** 2

Software Setup

- **Java version:** 1.8.0.144
- **JSON library org.json :** version 20180813

Data description

- **Total Size:** 960GB
- **Total Files:** 3985
- **Average File size:** 245MB
- **Average Number of lines in each file:** 100000
- **Size of each JSON string (one line):** 3024 Bytes

2 Task 1 : Data pre-processing

For the data pre-processing task, a simple java program was written to read files from HDFS and publish messages in kafka at a known broker and topic. The program published one tweet

at a time. The rate of input stream was **1429 tweets/second**. (The number of kafka messages produced per second).

The input stream was played for nearly 38 minutes. The major compute in the Data pre-processing were:

- Parsing json to object, normalizing the tweet (lowercase)
- Removing stop words and puncuations

The program was run for a window of 5 minutes, the amount of tweets that could be processed in the window is shown in the graph (Figure 1) plot below.

The average number of tweets processed per minute is **416.73 tweets/second** . The tweets which were processed to a 'csv' format were saved as text files in back in HDFS, it would serve as input for task 2.

The spark job was submitted with the following configuration:

- Number of executors = 4
- Number of cores = 4
- Memory = 8GB

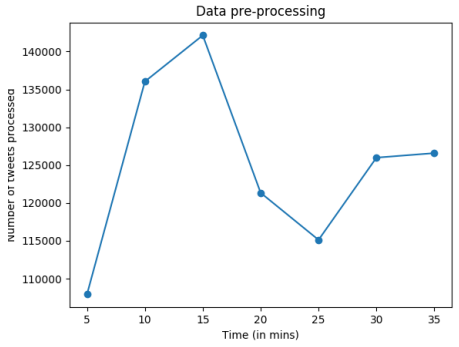

Figure 1: Data pre-processing

3 Task 2a : Top 5 trending hashtags

The computation of Top 5 trending hashtags take the csv format as input which are saved from the previous task. The rate of input stream was **852 tweets/second**. Here the window size is 90 seconds, the stats keep refreshing every 30 seconds.

The steps involved in the algorithm is summarized as follows :

- A pair Dstream entry with Hashtag as key and its count as value.
- The counter was incremented each time a hashtag featured in the tweet.
- A reverse mapping was which gave a Dstream with Count as key and Hashtag as value.

The final Dstream was saved to the HDFS. Every 90s, new batch of csv were considered. The number of tweets processed were **720 tweets/second**.

The spark job was submitted with the following configuration:

- Number of executors = 4
- Number of cores = 4
- Memory = 8GB

4 Task 2b : Distinct Approximate Count

In this task, we were asked to compute distinct approximate number of users using Flajolet-Martin algorithm in a given window of time, and accumulate that count over to the next window.

The input for this algorithm also was a csv just like the previous task. Every minute, the algorithm would report the number of distinct approximate users and save its state. In the next window of 1 minute, it would again execute the algorithm and compute the distict approximate counts, this time it would also add the count of the number of users that were saved previously.

A simple "modulo 1000000" hash function was used in this algorithm, it was sufficiently big enough for the number of csv tweets that were being streamed. (which is 905998). The hashcode was computed using the normal java hashcode function.

The steps involved in the algorithm is as follows :

- The userId was hashed and then a position 'pos' with respect to trailing number of zeroes was obtained. (eg: for binary string 1000, position is 3, for 1101, the position is 0)
- The above step was applied for each userId, and a maximum position was obtained. (max
- Once the stream for that window is exhausted, the number of distinct users is given by $2^{max(pos)}$

The spark job was submitted with the following configuration:

- Number of executors = 4
- Number of cores = 4
- Memory = 8GB

The number of users accumulated per hour is shown in the graph below (Figure 2) . X-axis is time in minutes, Y axis indicates the accumulated number of users per minute.

5 Task 3 : Sentiment Analysis

In this task, we were asked to compute the top 3 sentiments in a given window of time, the sentiments that are supposed to be identified are listed in the paper PANAS-t.

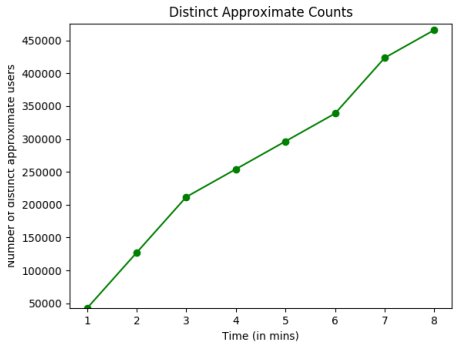

Figure 2: Flajolet-martin Algorithm

The window size was kept to be 3mins. The input stream rate was **1429 tweets/second**. The number of tweets process per second is **405.86 tweets/second**.

The algorithm can be summarized as follows:

- Each tweet json is parsed, the prepocessing steps of task 1 are applied.
- The descriptive words for each emotion mentioned in the PANAS-t is searched in the each tweet. And they are classified to a particular sentiment based on the presence of such words in either text of tweet or the hashtags.
- These sentiment count for a particular emotion is incremented whenever there is an appropriate match. At the end of each window, the top 3 sentiments are reported.

The case of words pointing to different sentiments are not considered since in the paper it has been mentioned that such instances are small and can be neglected.

The number tweets processed per 3 minute can be shown in the graph below (Figure 3). X-axis indicates time in mins, Y-axis indicates the number of tweets processed in a 3-minute window.

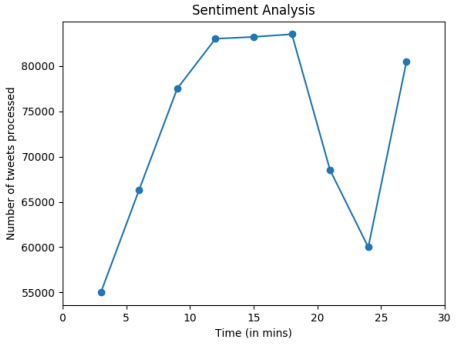

Figure 3: Sentiment Analysis Algorithm

Many of the tweets were in a different language, such tweets were categorized to be unknown emotion.

6 Task 4 : Weak Scaling

In terms of weak scaling, I introduced a delay of 1ms, 2ms and 4ms in the input stream rate. I ran the task 3 application with the following configuration, the memory per worker is maintained to be 8GB for all.

The spark job was submitted with the following configuration:

- Executors = 4 , Cores = 4, delay = 1ms, observed tweet rate/executor = 91.47
- Executors = 2 , Cores = 4, delay = 2ms, observed tweet rate/executor = 109.56
- Executors = 1 , Cores = 4, delay = 4ms, observed tweet rate/executor = 133.57

The number of tweets processed per single core over a single minute window is shown in the graph below (Figure 4):

The X-axis indicates the amount of delay between 2 kakfa produce messages, and the Y-axis indicates the number of messages processed per executor.

A total of 200000 tweets were played as a stream with 1ms delay, 2ms delay and 4ms delay respectively. For 1ms delay, stream rate was 431 tweets/second. For 2ms delay, stream rate was 300 tweets/second. For 4ms delay, stream rate was 184 tweets/second.

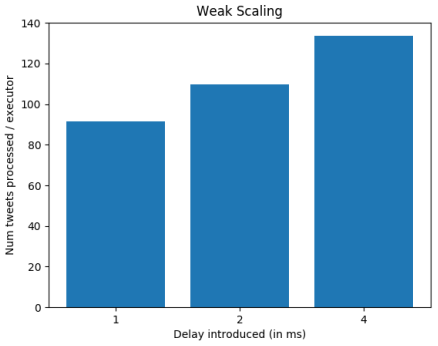

Figure 4: Weak Scaling Algorithm

From the observation we can say that the algorithm scales depending on the stream rate.

7 Acknowledgements

I would like to thank Swapnil for the Apache Spark-streaming primer to help kick-start the assignment.