DS 256 - Scalable Systems for Data Science
Assignment 0

Shriram R.
M Tech (CDS)
06-02-01-10-51-18-1-15763

January 20, 2019

1 Introduction

Analytics of Twitter data has been performed using Apache Spark through its Java interface. The experimental setup, results, plots and analysis are described in detail in the following sections.

2 Experimental Setup

2.1 Hardware

Experiments were performed on a commodity cluster having 24 compute nodes. Each node has a 8-core AMD Opteron 3380 processor clocked at 2.6Ghz along with 32GB RAM and 2TB HDD and runs Ubuntu 16.04 LTS (64 bit) with Linux 4.4.0-139-generic. The nodes are connected through a Gigabit Ethernet switch.

2.2 HDFS and Spark

The HDFS environment has a capacity of 30.93TB with block size 128MB, replication factor of 2 and heartbeat delay of 3s. The global configuration of Spark for all experiments is 4 executors (containers) each having 4 cores and 8GB of memory. The Driver memory was set to 512MB. Apache YARN was used to coordinate the job execution. Experiment specific detail if any is provided below.

2.3 Dataset

The dataset used is a collection of tweets from Twitter between 17th Oct - 16th Dec 2016. The total dataset size is about 955GB and is stored in HDFS by partitioning the dataset into 3984 files of comparable size. Each tweet is described in Twitter Tweet JSON format and is of size approximately 5-10KB . The structure of JSON is described in [1].
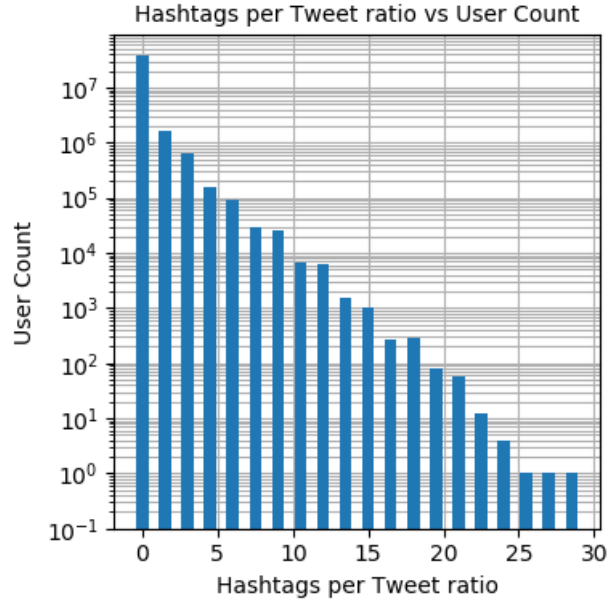
3 Histogram

The pseudocode for the histogram of hashtags per tweet is given below,

```
      // Load text file into RDD
   1. JavaRDD twitterData = textFile(inputFile)
      // Parse JSON and get the hash count along with User ID
   2. JavaPairRDD hashCount = twitterData.flatMapToPair(getHashCount())
      // Get the average hashtag per tweet for each user
   3. JavaRDD avgPerUser = hashCount.reduceByKey(x, y -> x+y).map(sum/count)
      // Get the bucket values and frequencies for the given bucket count
```

```
    4. histogram = avgPerUser.histogram(bucketCount)
    5. Write histogram to HDFS
```

The job was executed for the full dataset using the global configuration. The job took about 2.9 hours to complete. The job initially sustained failed stages which took about 4.5 hours. This occurred due to failure in metadata fetch. The job generated about 17.9GB of Shuffle data. Bucket count was chosen arbitrarily as 20 and the result is plotted below,

It can be observed that the user count decreases as the Hashtag per tweet ratio increases. About 93.5% of the users fall in the first bucket. The maximum Hashtag per tweet ratio in the given dataset is about 28.5 and the average value is about 0.16.
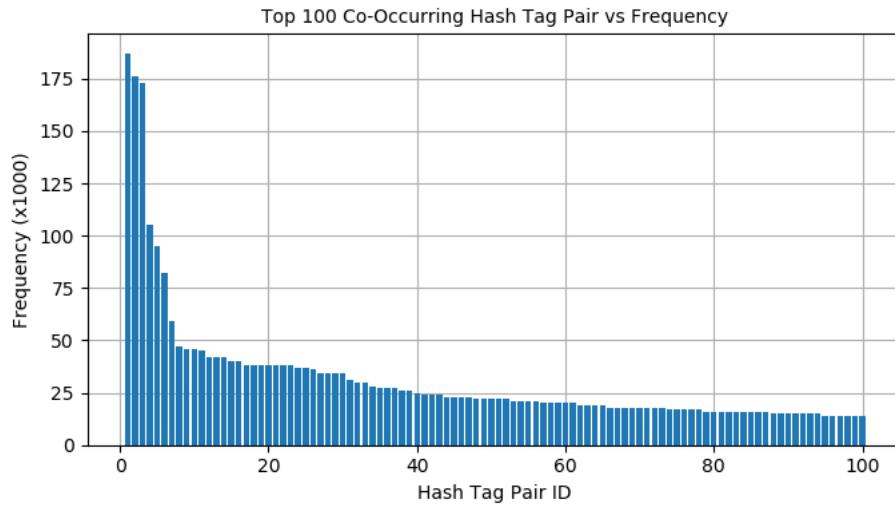
4 Co-occurring Hashtags

The pseudocode for finding the top 100 frequently co-occurring hashtags is given below,

```
    // Load text file into RDD
 1. JavaRDD twitterData = textFile(inputFile)
    // Parse JSON and get the hash tag pairs available in each tweet
 2. JavaPairRDD hashTags = twitterData.flatMapToPair(getHashTags())
    // Get the count for each distinct hash tag pair, sort descending by value
 3. JavaRDD hashCount = hashTags.reduceByKey(x, y -> x+y).mapToPair(swap key-value)
                                                          .sortByKey(false)
    // Get top 100 frequent hash tag pairs
 4. pairs = hashCount.take(100)
 5. Write histogram to HDFS
```

The job was executed for the full dataset using the global configuration. The job took about 2.8 hours to complete and generated 9.4GB of Shuffle data. The list of top 100 pairs of Hashtags is available in Appendix A. The frequency of occurrence od these pairs is plotted below,

Top 100 Co-Occurring Hash Tag Pair vs Frequency

The Hash Tag Pair IDs are alloted in the decreasing order of frequency. It can be observed that the frequency drops exponentially along the x-axis. The most frequent pair of Hash Tags was present in about 187,000 tweets. The Hash tags were from different languages.

5   Temporal Interaction Graph

The pseudocode for getting the temporal interaction graph is given below,

```
// Load text file into RDD
1. JavaRDD twitterData = textFile(inputFile)
// Parse JSON and get the hash count along with User ID
2. JavaPairRDD hashCount = twitterData.flatMapToPair(getHashCount())
// Get the average hashtag per tweet for each user
3. JavaRDD avgPerUser = hashCount.reduceByKey(x, y -> x+y).map(sum/count)
// Get the bucket values and frequencies for the given bucket count
4. histogram = avgPerUser.histogram(bucketCount)
5. Write histogram to HDFS
```

The job was executed only on 1% of the dataset. The job took a total of 9.1 minutes to complete and generated about 180MB of Shuffle data. Execution on larger dataset sizes resulted in job failures due to insufficient memory and/or workers getting disconnected and in some cases few tasks that were running indefinitely.

6   References

2. DS 221 Course lecture notes

7 Appendix - A

Hash Tag Pair ID	Hash Tag Pair
1	gameinsight,androidgames
2	androidgames,android
3	gameinsight,android
4	????????,?????????
5	?????,BTS
6	???,GOT7
7	?????,?????
8	??,JIMIN
9	????,TWICE
10	??,?????
11	?????,?????
12	???????,???????
13	???????,?????????
14	????????,???????
15	np,SoundCloud
16	??,?
17	??????,???????
18	???????,??????
19	???????????,??????
20	???????????,???????
21	???????,???????????
22	?????????,??????
23	?????????,???????????
24	??????,????
25	job,Hiring
26	?,Hadith
27	??,?????
28	?,?????
29	??,?????
30	?????,?????
31	?????,MONSTA_X
32	??,SUGA
33	?????,JIMIN
34	??,?????
35	?,???_?????
36	MTVStarsFifthHarmony,AP40FifthHarmony
37	????,GOT7
38	???,SEVENTEEN
39	MTVStars5SOS,AP405SOS
40	?????,SUGA
41	GOT7,BamBam
42	??,V
43	??,VIXX
44	Hiring,CareerArc
45	??,GOT7
46	?????,?????
47	EMABiggestFansBeyonce,ARIASBEYONCE
48	Necklace,Jewelry
49	?????,V
50	??,JUNGKOOK
51	JIMIN,BTS
52	V,BTS
53	?,V
54	KCAArgentina,5SOS
55	job,CareerArc
56	JB,GOT7