BIG-DATA CONTENT RETRIEVAL, STORAGE AND ANALYSIS FOUNDATIONS OF DATA-INTENSIVE COMPUTING

VIVEKANANDH VEL RATHINAM, JERRY ANTONY AJAY
AFFILIATION: CSE587, UNIVERSITY AT BUFFALO, vvelrath@buffalo.edu , jerryant@buffalo.edu

1. Abstract

This project deals with using Hadoop framework to analyze twitter data. Using Hadoop, the present trends in twitter is gathered. Some of the statistics we collected include the total count of hashtags, collection of users and their followers, and finally the most number of recurring words in a collection of tweets. Some of the fetched data is also visualized and interesting patterns are obtained in R.

2. Project Objectives

- To understand the components and core technologies related to content retrieval, storage and data-intensive computing analysis.
- To explore designing and implementing data-intensive (Big-data) computing solutions using MapReduce (MR) programming model.
- Using Hadoop 2 for HDFS and MR infrastructure.
- To visualize the data using appropriate tools.
- Create a detailed documentation of the experience.

3. Project Approach

We worked on the twitter4j API to fetch relevant statistics in-order to process them. The procedure of gathering the data extended over a period of days until we were satisfied that we had a collection of a small BIG DATA. Two types of data was collected in this approach. Firstly, a raw file containing around 200,000 tweets in English. Secondly, a file containing usernames and the total number of followers for every user.

As the first step toward analyzing the tweets collected over a period of time, we sorted the tweets into three files where the files had a collection of all hashtags and the count, usernames and the count, and finally the collection of words encountered while reading the tweets and their count. This process was accomplished using MR on Hadoop framework. This information is then used to identify the most trending hashtags, most active users and users with most followers respectively.

The second part of the project involves identifying co-occurring hashtags using pairs and stripes approach. The pairs approach was handled in MR by dividing the total number of occurrences of a pair to its marginal. The stripes approach involved aggregating the key value pairs and letting the MR engine to automatically sort the input and distribute the keys to the reducers internally.

The third part of the project involved developing K-means clusters to cluster users based on the total number of followers they had. The internal counters of Hadoop was used to compute the centroids. This process is repeated until convergence. The total number of centroids was assumed to be three and the iterations started with initial values of 50, 500 and 1000 respectively.

Finally, the fourth part of the project involved finding the shortest path of the most popular hash tag. This involved developing a connected network of sender -> receiver pattern and labelling the network for determining the shortest path between the nodes of this network. The job is given to an MR engine that finds the shortest path iteratively.

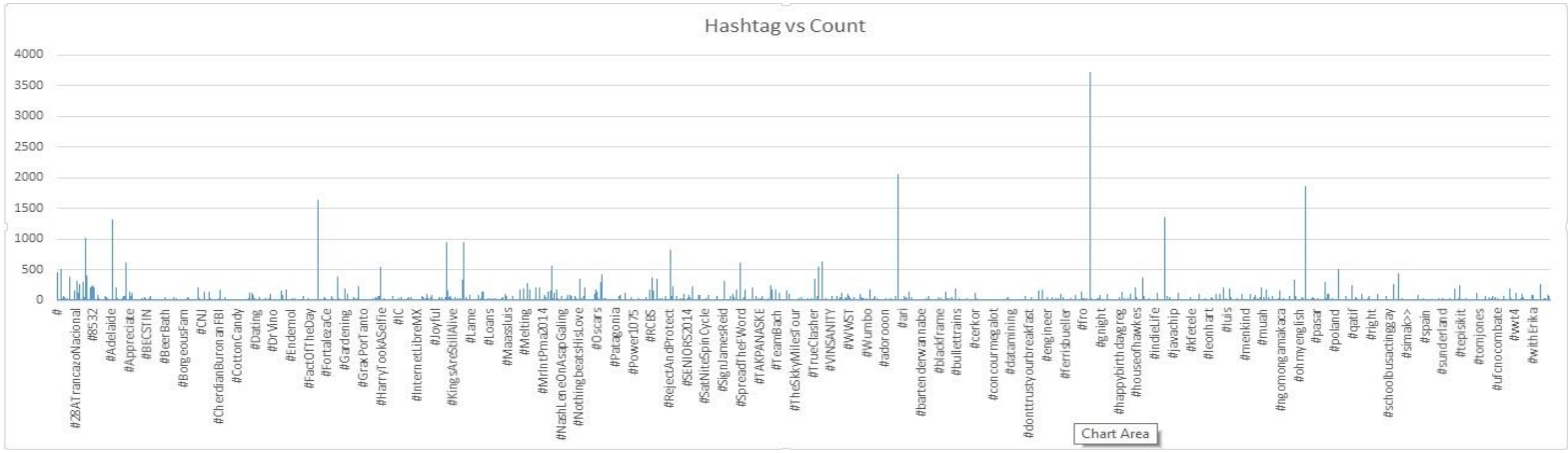
4. Word Count

The word count phase involves collecting all the tweets that gets streamed into our file over a period of time. We have collected around 250MB of tweets which approximately correspond to 500,000 tweets; a small BIG DATA collection. The MR framework is put into use in populating three files that have the word count, hashtag counts and user counts respectively. The pseudo-code for populating the files are given below. The mapper processes every word that enters into it and attaches a count of 1. The words are then sent to the appropriate reducers where the count of each word is aggregated.

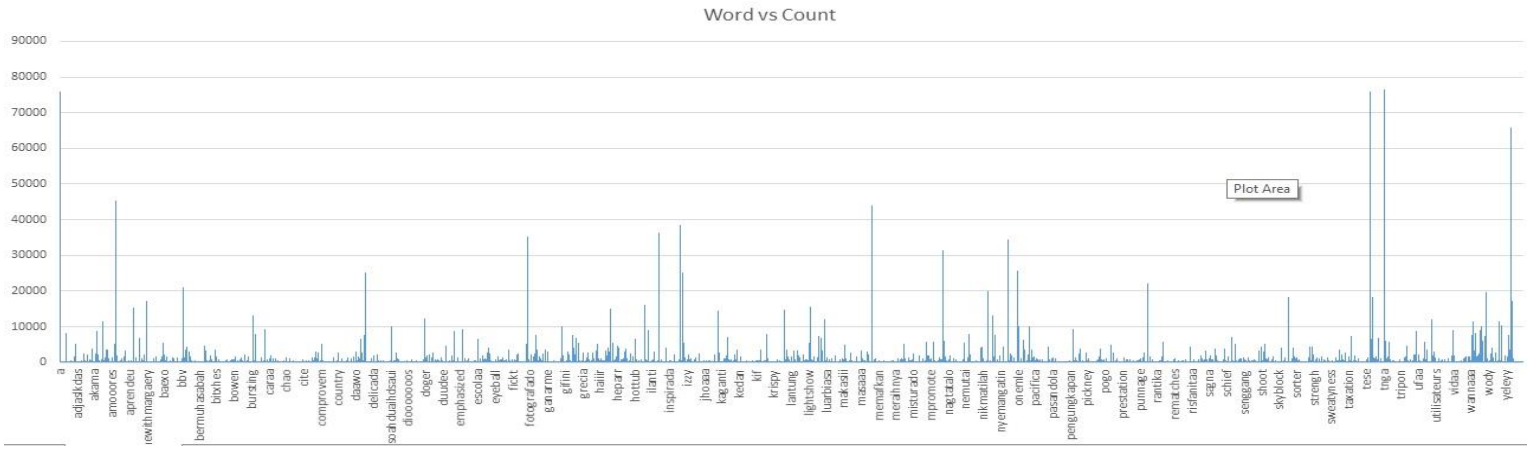
Pseudo-code:

Mapper:
For(every word inputted)
  Write to reducer(word, count as 1)

Reducer:
For(every word inputted){
  If multiple instances exists, then, add it to the aggregate sum
   Write to output(word, aggregate sum)
}



From the above graph, we observe that the hashtag "#gameinsight" was the most trending topic. Some of the other popular topics include "#android", "#openflow","#FilmTANIA" and "#AdminDirectPopularPENIPU"



From the above graph, we observe that the most popular words are usually the pronouns and words such as ‘the’ and ‘and’. Excluding such participles, we have some of the highest tending words that include ‘like’, ‘love’,’no’,’not’,’people’,‘all’, etc.

5. a.Co-occurrence using stripes

The Co-occurrence problem using stripes approach involves populating the Hashmap with appropriate values in the Mapper and then using this information to display the co-occurrence of two words back to the user. The stripes method is used to find the co-occurrence values. In this method, the HashMap is populated as the word as the key and all the words that co-occur along with it as values. This HashMap is then sent to the reducer which sets the frequency for each word. The frequency is populated by dividing the sum of occurrence by the marginal. The pseudo-code written below provides a clearer explanation of the theory involved.

Pseudo-code:

Mapper:

Input: ‘token’ as HashMap

```
For (every token i until end of HashMap){
  Extract string ‘word1’ from token.
  Initialize MapWritable map
  For (every token j= i+1 until end of HashMap){
      Extract string ‘word2’ from token if exists.
       If(word2 exists)
          Increment count value associated with word2
       Else
          Initialize count value of word2 to 1.
  }
}
```

Reducer:

Input: Map of words and count

Compute marginal for each word in the hashmap by taking it's count.
For(every 'word' in the value of the Map){
    String 'word_pair' = Concatenation of the key and 'word' of the map
    Set key of Map to 'word_pair'
    Frequency = value/marginal
    Set value of Map to 'Frequency'
}

b. Co-occurrence using pairs

The Co-occurrence problem using pairs approach involves computing two values for every single occurrence. The first being a word_marginal which is a concatenation of the word itself and an "*" and secondly, a word_pair, which is a concatenation of the word and the co-occurring word. These two values are then sent to the reducer. The reducer then adds to the variable named 'marginal' all the pairs that end with "*" and computes the frequency of the word pair by dividing the number of occurrences of the combination by the marginal. The code is self explanatory.

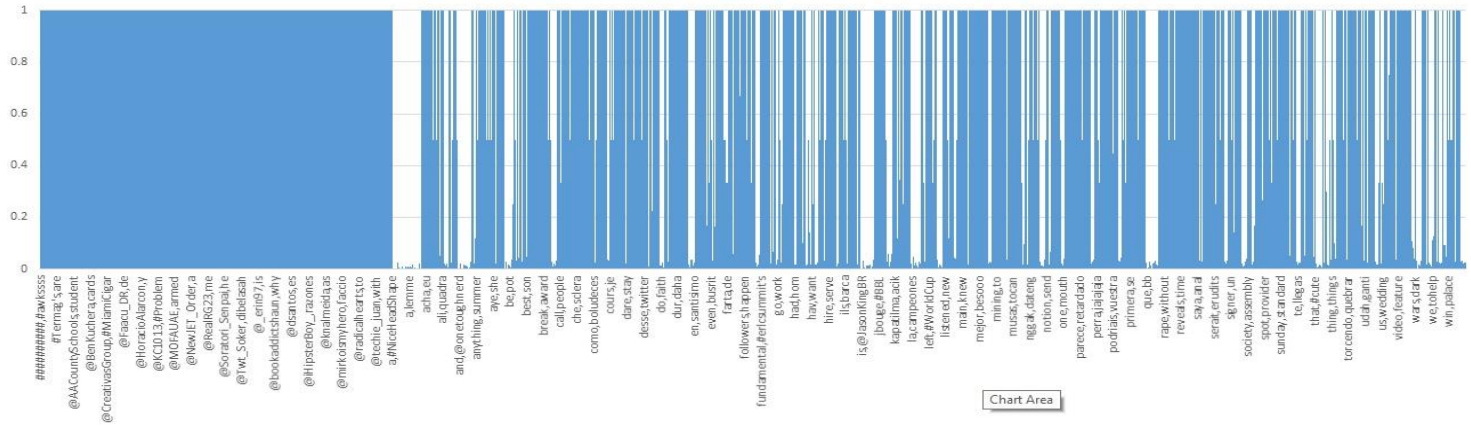
Pseudo-code:

Mapper:
Input: 'token' as ArrayList

For (every token i until end of ArrayList){
  Set 'word_marginal' = Concat(word[i],"*")
  For (every token j= i+1 until end of ArrayList){
      Set 'word_pair' = Concat(word[i],word[j])
      Output to reduce: 'word_marginal' and 'word_pair'
  }
}

Reducer:
Input: Text key and Int values
For(every 'key' ending with "*")
  Add the count to 'marginal'

For(every other combination){
    Frequency = No. of occurrences of the combination/marginal
    Set value to 'Frequency'
}
Write the combination(key,value) to output



We observe from the above graph that the many values have co-occurrence value of value. This is because the combination of these hash-tag is unique. Some of these include: "cushions,#class", "mainstreet,brave"and "spot, provider". The most common ones have a low co-occurrence value. Some of the values include,"and,a", "by,large", "but,it", etc.

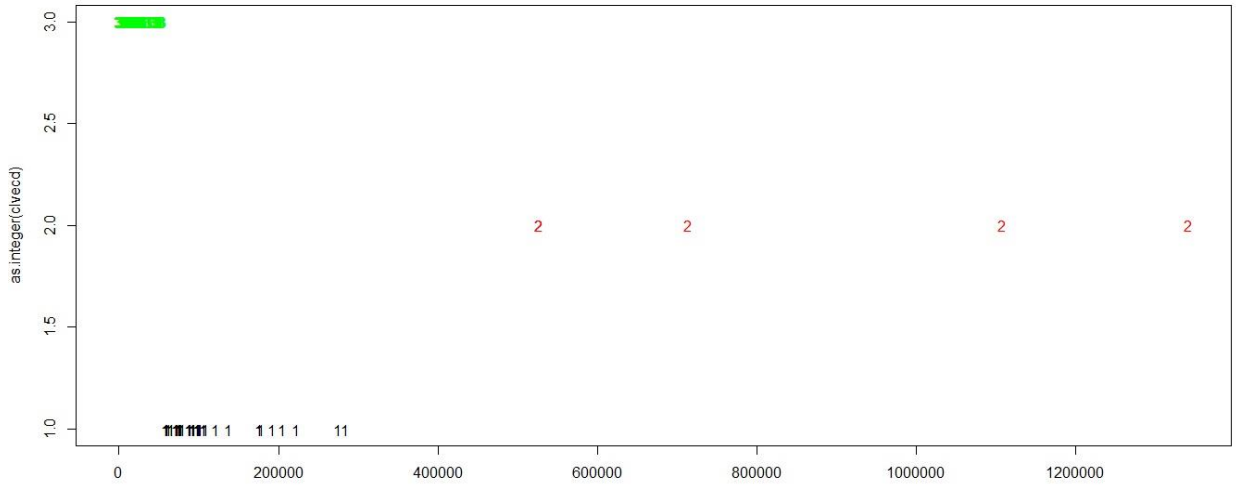
6. K-means

We have used the popular K-means method on a MR framework. The mapper firstly initializes the cluster centroids to 50, 500 and 1500 respectively. The number of centroid values are taken to be three. The values of these centroids are stored in the internal counter of Hadoop and can be accessed at the end of every run. The Mapper initially puts the users to the respective clusters based on the centroids and the reducer updates the new centroid by the logic shown in the code below.

Pseudo-code:

Mapper:
Initialize the first cluster to 50, second to 500 and third to 1500
For(Every entry in the input){
  Extract the 'user' and 'follower count'
  If('user' fits into any of the clusters){
    Add user to the respective cluster
  }
}

Reducer:
Input: Users and values belonging to clusters
For(All the values associated with a cluster){
  Sum+=values
  Count++
}
new_centroid = sum/count
Set First, Second and Third centroid accordingly



In the above representation generated by R we find that the values on the x-axis which symbolizes the number of followers have been grouped accordingly into three different clusters. The first cluster contains values ranging between 100000 -300000, cluster 2 contains values between 500000-onwards and cluster three contains values from 0 to 100000. Running this example in R gave us a value the frist centroid as 1217, second centroid as 311643 and the third centroid as 1338638.

7. Shortest-path problem

The Shortest-path problem aims at identifying the shortest path from a node to a destination in a graph. This mechanism is achieved in a MR framework using the pseudo-code listed in the section below. The Mapper reads the input graph and for every node, reads it's adjacency list and updates it's value. This huge collection of possibly duplicate keys are sent to the reducer which outputs the minimum distance value of every single token to the user.

Pseudo-code:
Mapper:

```
While(tokens present in input file){
    Extract the distance and the adjacency list for each node
    For each adjacent node, add the cost of reaching the present node to the adjacent node
     Emit to reducer(token, distance)
}
```

Reducer:
Input: Distance graph from Mapper

```
While(tokens present in input from Mapper){
    Extract the least value of the distance of every single token
    Emit to user(token, distance)
}
```

8.References:

1. Doing Data Sciences
2. Blog: http://codingjunkie.net/cooccurrence/