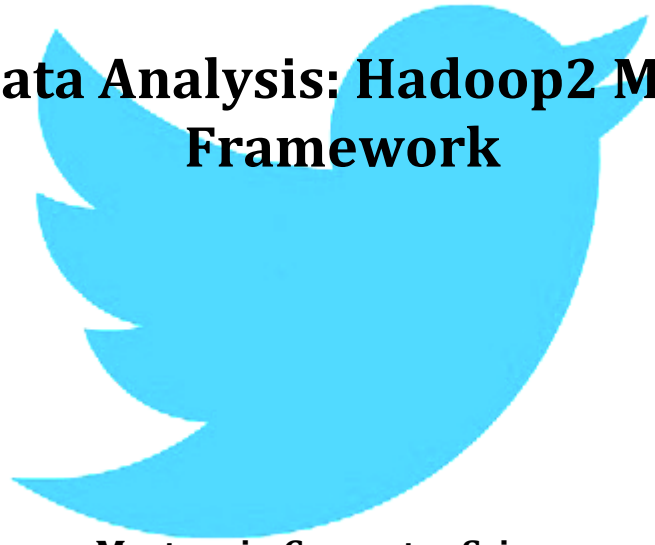
Data Intensive Computing
CSE 487/587
Project Report 2

Twitter Data Analysis: Hadoop2 Map Reduce Framework



Masters in Computer Science
University at Buffalo

Submitted By
Prachi Gokhale (prachich@buffalo.edu)
Pradnya Kulkarni (pradnyak@buffalo.edu)

Website:http://www.acsu.buffalo.edu/~prachich
http://www.acsu.buffalo.edu/~pradnyak

1. Abstract:

Social networking sites nowadays are contributing a lot towards big data. In order to find the interesting patterns or trends from this huge data, data scientists need to clean, integrate, aggregate and analyze the data. The purpose of this project is to find out trends by aggregating the data in social networking site such as Twitter.

2. Project objectives:

The core objective of the project is to understand the components and core technologies related to content retrieval, storage and data intensive analysis of large corpus of data collected over a specific period of time. One of the important objectives is to demonstrate the analysis using visualization tools.

Content Retrieval: The large amount of data is collected using java Twitter streaming API.

Data Processing: Data collected over a period of time is processed by using parallel and distributed processing software framework developed by Apache Hadoop and using map reduce programming model.

Storage: This data is stored in a certain format so as to form key value pair which is needed to feed to mapper in map-reduce programming approach. The data is stored in Hadoop2 Distributed File System.

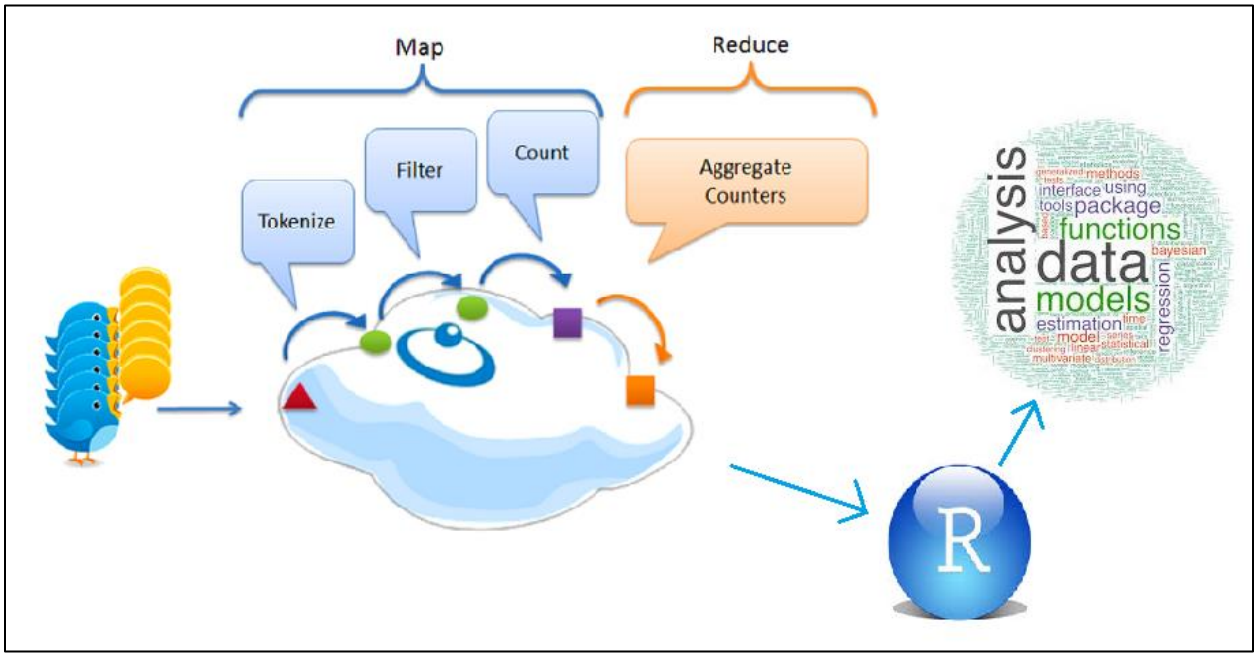
Data Analysis: The output obtained from reducer phase is analyzed using a data analysis tool like RStudio in language R.

Visualization: Various ongoing trends on social networking sites are aesthetically represented using plotting libraries in R.

3. Project Approach:

1. Studying Hadoop 2.0 architecture
2. Configuring Hadoop on the machine.
   The installation guide is:
   https://drive.google.com/file/d/0BweVwq32koypbjd6T19QWmhUZlU/edit?usp=sharing
3. Understanding map reduce functionality
4. Implementing simple word count program in Map Reduce
5. Studying Twitter API

Workflow:



4. Data

A. Data format and source

1. Data is aggregated from Twitter using java Twitter Streaming API
2. Tweets for the current trending topic "Indian Premier League" are collected using filter query in Twitter API. Various hash tags for IPL are added as filter while collecting data.
3. Data is collected for different ranges of dates (day-range ,week-range)
4. Aggregated raw data is cleansed to some extend before analyzing it using Map-Reduce methods.
    - Removal of stop words
    - Removal of hyperlinks
    - Removal of punctuations
5. The format of collected tweets has following fields:
    - Date
    - User name
    - Follower count of the user
    - Tweet text
6. Each tweet is separated by new line and the above information is separated by a delimiter so that the processing in Map reduce is easy.

Sample data screenshot:

```
Wed_Apr_23_12:48:09_EDT_2014 *** FRANK *** 547 *** Ishwar Pandey Vs Sanju Samson   Future Indian cricket safe hands  #IPL #RR ***  $$
Wed_Apr_23_12:48:11_EDT_2014 *** CRICKET%x%Kibow *** 2 *** On Now: Chennai Super Kings vs Rajasthan Royals #cricket #Yahoo ***  $$
Wed_Apr_23_12:48:11_EDT_2014 *** p.saimohanreddy *** 1 *** I m backing #Ashwin @IPL #playerbattles  Vote your choice now #PepsiIPL7 ***  $$
Wed_Apr_23_12:48:12_EDT_2014 *** Kapil%Sharma *** 31 *** very gud stuff @msdhoni   #PepsiIPL ***  $$
Wed_Apr_23_12:48:13_EDT_2014 *** Jaideep%Chakrabarty *** 130 *** Dhoni clearly throwing challenge Sanju going up 2 stumps  Come dude    #IPL7 ***  $$
Wed_Apr_23_12:48:15_EDT_2014 *** Cricket%Live%Score *** 77 *** Yorkshire v Northamptonshire Leeds   Yorkshire won an innings 120 runs #cricket ***  $$
Wed_Apr_23_12:48:17_EDT_2014 *** John%Venturi *** 12 *** Just You  Enter chance #win  15 00 points  Play #Sweeps via @SYWSweeps ***  $$
Wed_Apr_23_12:48:20_EDT_2014 *** ABHISHEK *** 360 *** @sanjusamson8 has far faced an over dot balls #RRvsCSK #IPL7 ***  $$
Wed_Apr_23_12:48:20_EDT_2014 *** gayatri%vishwanath *** 102 *** RT @ChennaiIPL: 4  Now need 46  #whistlepodu #csk ***  $$
Wed_Apr_23_12:48:21_EDT_2014 *** Rajamanickam *** 0 *** Come #CSK ***  $$
Wed_Apr_23_12:48:23_EDT_2014 *** IPLLive_Ckt *** 256 *** Pepsi_IPL 10TH_MaTcH CSK_vs_RR TraGeT_141 RR_17 1 Over_3 1 S_SaMsoN_0  A_RahaNe_11   #CSKvRR #PepsiIPL Follow
@IPLlive_Score ***  $$
Wed_Apr_23_12:48:24_EDT_2014 *** Joel%Norrod *** 0 *** Covet-Worthy  Enter chance #win  10 00 points  Play #Sweeps via @SYWSweeps ***  $$
Wed_Apr_23_12:48:27_EDT_2014 *** Sze%Cheung *** 3 *** Pick up Steam  Enter chance #win  5 00 points  Play #Sweeps via @SYWSweeps ***  $$
Wed_Apr_23_12:48:30_EDT_2014 *** Ram%Charan%Tej%D *** 0 *** Come ishwar pandey great playaa   never ever lose your hope    #CSK  Following Match 10 IPL 2014 ***  $$
Wed_Apr_23_12:48:31_EDT_2014 *** Rajasthan%Royals *** 264775 *** Sanju off mark well  Make partnership count #Royals   #WeAreTheRoyals #RR ***  $$
Wed_Apr_23_12:48:32_EDT_2014 *** Niraj%Bayla *** 84 *** Whatt class keeping Dhoni   just amaziing    :D #RRvsCSK #IPL ***  $$
Wed_Apr_23_12:48:32_EDT_2014 *** Prabhu *** 1460 *** Some Justice Mudgal wondering Target #CSK set #RR today #CSKvsRR ***  $$
Wed_Apr_23_12:48:32_EDT_2014 *** Pakistan%Cricket *** 2257 *** IPL T20: Super Kings score 140 6 Royals: Chennai Super Kings put up total 140    #ipl #ipl2014 #ipl7 ***
 $$
```

B. Amount of data collected and details

1. Data is collected for different ranges of dates (day-range , week-range)
2. Total of 300 MB data is collected over the period from 17th April to 26th April.

Total number of tweets collected -

Date	Number of Tweets
17th April	734046
18th April	1018782
19th April	207808
20th April	235889
21th April	106554
22th April	120003
23th April	63990
24th April	355142
25th April	311743
26th April	293255
Total	**3447212**

5. Analyzing output data and creating visualizations

The output obtained from reducer is sorted by using Unix utilities like grep and sort.

R is used to aggregate and to analyze the data output from Map Reduce programming model. The output of reducer is converted into CSV format and read using R. Various graphs are plotted so as to showcase the popular trends on twitter.

Following R libraries are used to create plots, graphs and summaries as a part of visualization.

```
library("sqldf")
library("ggplot2")
library("openNLP")
library("wordcloud")
library("tm")
library("twitteR")
library("plotrix")
library("doBy")
```

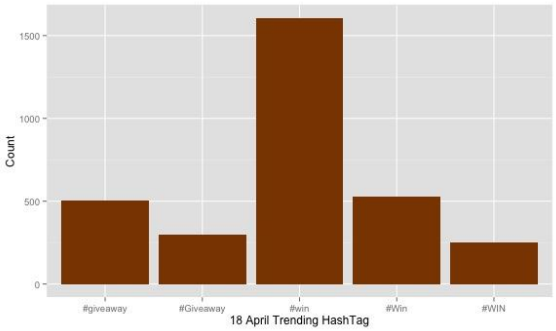
6. Problem Statement , Implementation and Analysis

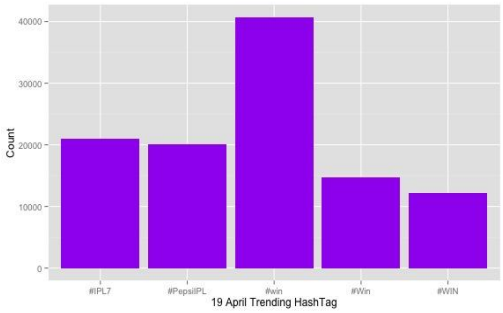
A. Design and implementation of MR workflow to extract various information from twitter data.

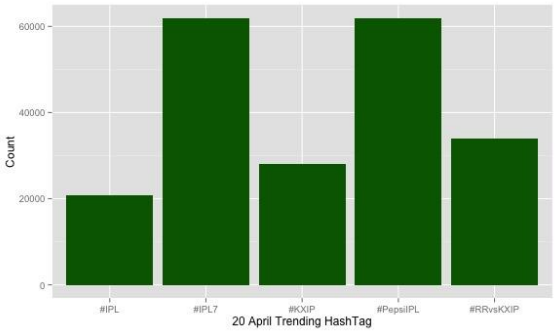
a. Word Count: We formed a corpus of tweet text collected using Twitter API and formed a word cloud.

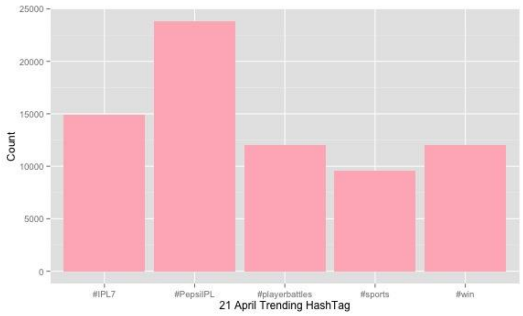
b. Most Active Users: We gathered usernames from twitter, performed aggrgatation and formed a cloud of active users on twitter.
The following cloud represents top 100 active users on twitter.

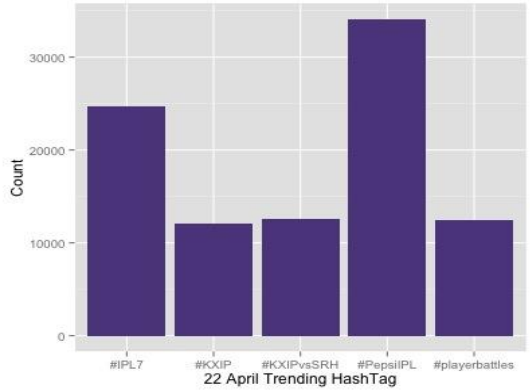
c. Hash Tags: The following plots show day-wise (from 18th April to 25th April 2014) trending hash tags with their count.

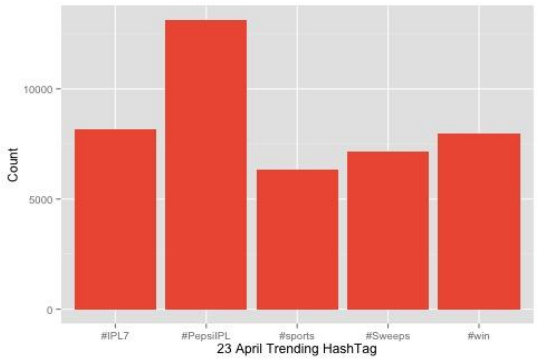


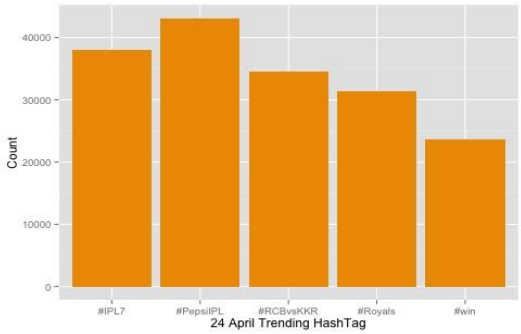


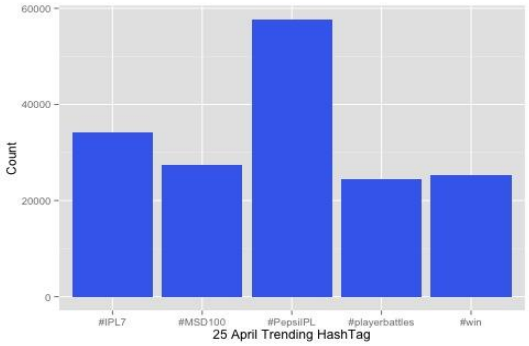


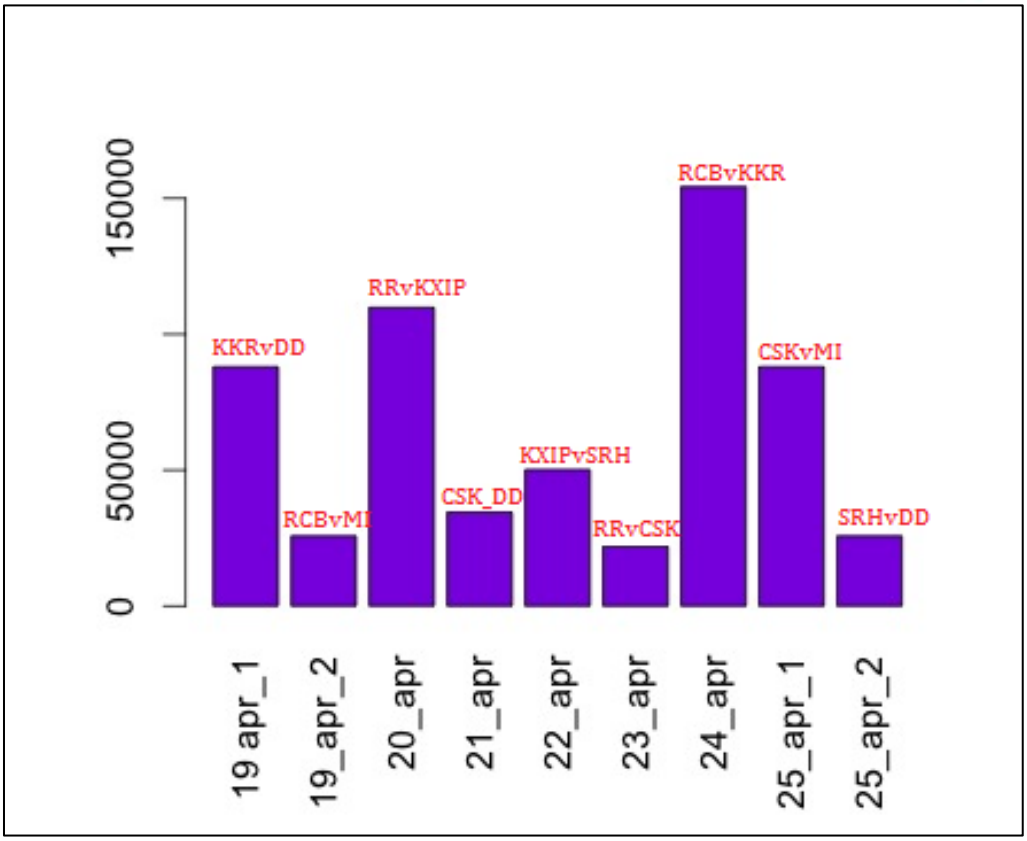








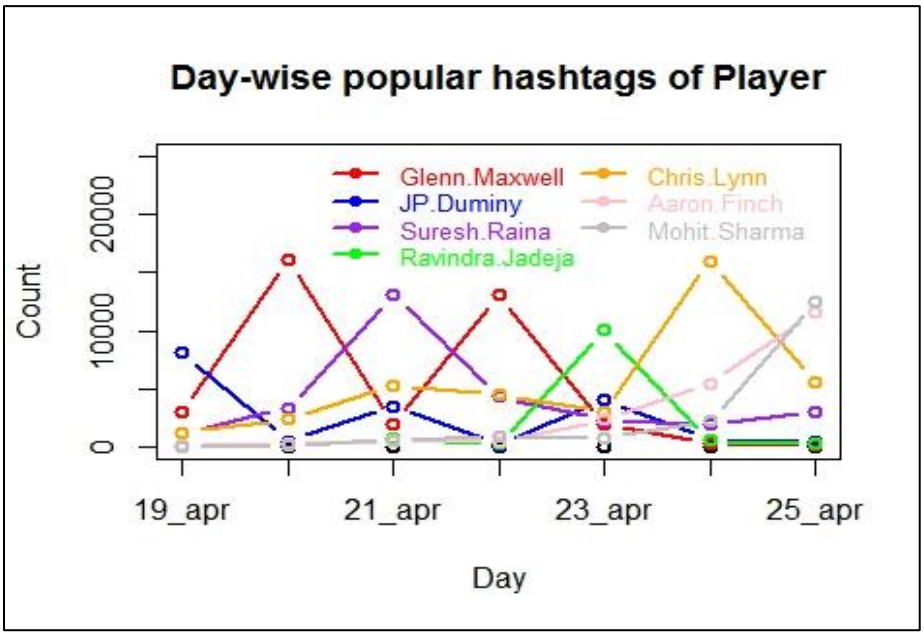




The above graph shows comparison of highly occurring hash tag counts from 19th April 2014 to 25th April 2014. On 19th and 25th April two matches played as shown in plot.
Below is the detailed analysis of each match depending on collection of their hash tags.

Date	Team	Winner	Popular #Tags
19th April 2014	Royal Challengers Bangalore vs Mumbai Indian	Royal Challengers Bangalore	#RCB ,#RCBvMI,#MI,# RCBLive
19th April 2014	Kolkata Knight Riders vs Delhi Daredevils	Delhi Daredevils	#KKRvDD, #KKR
20th April 2014	Rajasthan Royals vs Kings XI Punjab	Kings XI Punjab	#KXIP, #RR, #RRvKXIP
21st April 2014	Chennai Super Kings vs Delhi Daredevils	Chennai Supper Kings	#CSKvDD, #DDvCSK, #CSK
22nd April 2014	Kings XI Punjab vs SunRisers Hyderabad	Kings XI Punjab	#KXIPvSRH, #SRHvsKXIP
23rd April 2014	Rajasthan Royals vs Chennai Super Kings	Chennai Super Kings	#RRvCSK,
24th April 2014	Royal Challengers Bangalore vs Kolkata Knight Riders	Kolkata Knight Riders	#KKR, #RCBvKKR , KKRvRCB
25th April 2014	SunRisers Hyderabad vs Delhi Daredevils	SunRisers Hyderabad	#SRHvDD
25th April 2014	Chennai Super Kings vs Mumbai Indians	Chennai Super Kings	#CSKvMI, #CSK, #MIvCSK

We also performed player wise EDA on hash tags and graph of count of hash tags of famous players over this league is shown below:



As per the stats, if player has performed well on a specific date, then people tend to tweet more about it on social networking sites hence more count.
Below is detailed analysis of remarkable players on collection of their day-wise hash tags:

Date	Teams	Remarks	#Tags	Player Name
19th April 2014	Royal Challengers Bangalore vs Mumbai Indians	Parthiv's fifty in a low scoring match	#partivpatel	Parthiv Patel
19th April 2014	Kolkata Knight Riders vs Delhi Daredevils	Duminy's half century in the winning cause	#DUMINI, #dumini	JP Duminy
20th April 2014	Rajasthan Royals vs Kings XI Punjab	Maxwell Again batted with a grit to grind RR bowlers	#Maxwell	Glenn Maxwell
21st April 2014	Chennai Super Kings vs Delhi Daredevils	Raina's 50 has set the tone for Chennai	#Raina , #raina	Suresh Raina
22nd April 2014	Kings XI Punjab vs SunRisers Hyderabad	43 balls 95 again for the third consecutive Man of the Match award	#Maxwell, #MaxwellMillerMagic	Glenn Maxwell
23rd April 2014	Rajasthan Royals vs Chennai Super Kings	Rajasthan's batting crumbled against spin of Jadeja	#jadeja	Ravindra Jadeja
24th April 2014	Royal Challengers Bangalore vs Kolkata Knight Riders	Superb catch by Chris Lynn sealed the improbable win for KKR	#ChrisLynn, #Lynn	Chris Lynn
25th April 2014	SunRisers Hyderabad vs Delhi Daredevils	Aaron Finch propelled Hyderabad to a BIG score	#AaronFinch	Aaron Finch
25th April 2014	Chennai Super Kings vs Mumbai Indians	4 Wickets in 14 runs tied Mumbai to the low score	#MohitSharma, #mohitsharma	Mohit Sharma

A Custom Partitioner is user to separate the hash tag , usernames and words from tweet text.

Following is the code snippet for the Custom Partitioner class.

CustomPartitioner.java

```
package sample;


import org.apache.hadoop.io.IntWritable;

public class CustomPartitioner extends Partitioner<Text, IntWritable> {

    public int getPartition(Text key, IntWritable value, int numReduceTask) {
        // TODO Auto-generated method stub


        String term1 = key.toString();
        if(term1.startsWith("#") && term1 !=null)
        {
            return 3%numReduceTask;
        }
        else if(term1.startsWith("@") && term1 !=null)
        {
            return 2%numReduceTask;
        }
        else if(term1 !=null)
        {
            return 1%numReduceTask;
        }
        return 1;

    }
}
```

B. Extracting co-occurring hash tags using “pairs” and “stripes” approach.

a. Co-occurring Hash Tags Pair Approach
b. Co-occurring Hash Tags pair-wise with Relative Frequency
c. Co-occurring Hash Tags Stripe Approach
d. Co-occurring Hash Tags stripe-wise with Relative Frequency

Two approaches are used to find the co-occurring hash tags

a. Pair Approach

Algorithm:

```
Class Mapper
    method Map(docid a; doc d)
    Emit((hash t , hash p); count of occurrence )

Class Reducer
  method Reduce(((hash t , hash p); count of occurrence)

    sum <- 0
    cnt <- 0
    for all pairs (hash t , hash p) , integers [r1; r2; : : :] do
    sum  <- sum + r
    cnt  <- cnt + 1
    ravg <-  sum=cnt;  (calculating relative frequency)
    Emit(string t; integer ravg)
```

b. Stripe Approach

Algorithm:

```
Class Mapper
    method Map(docid a; doc d)
    Emit(hash t; Map(co-occurring hash tags with hash t, count of
    occurence)

Class Reducer
  method Reduce(string t; Map(co-occurring hash tags with hash t ,
  count of occurence)

    sum <- 0
    cnt <- 0
    for all integer r , integers [r1; r2; : : :] do
    sum  <- sum + r
    cnt  <- cnt + 1
    ravg <-  sum=cnt;  (calculating relative frequency)
    Emit(string t; integer ravg)
```

Mapper of Stripes approach

TokenizerMapper.java | IntSumReducer.java

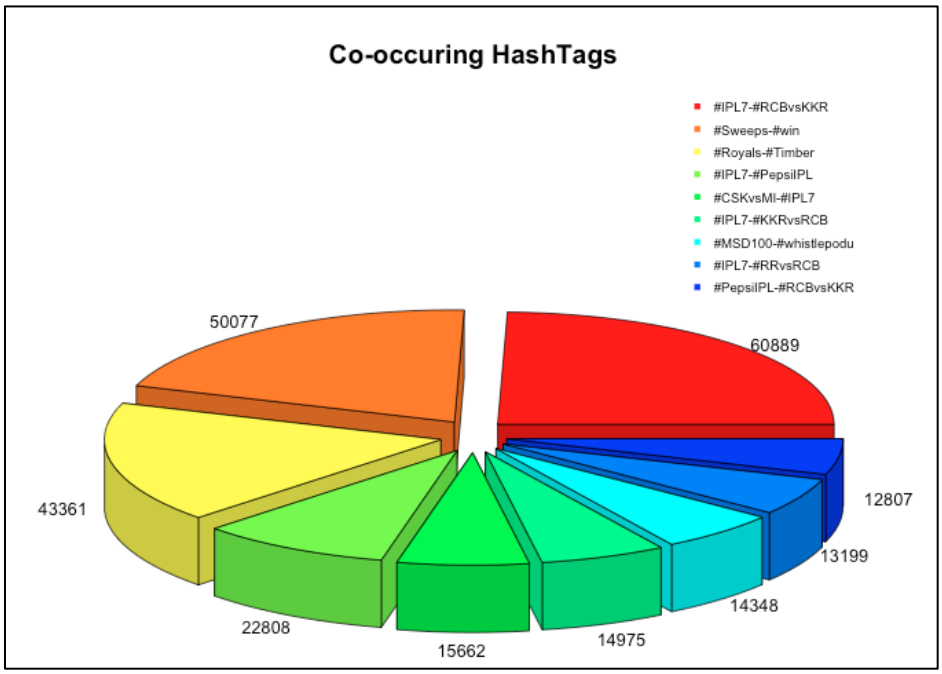
```
        {
            for (Entry<Writable, Writable> entry : all_hash.entrySet())
            {
                String token1[]= str_arr[i].split("\\*\\*\\*");
                if(token1.length >= 3 )
                {
                    String tokens [] = token1[3].split(" ");

                    for( int j =0 ; j<tokens.length ; j++)
                    {
                        if (tokens[j].startsWith("#"))
                        {
                            if(individual_hash.containsKey(new Text(tokens[j])))
                            {
                                int a = Integer.parseInt(individual_hash.get(new Text(tokens[j])).toS

                                IntWritable sum = new IntWritable(a);
                                    individual_hash.put(new Text(tokens[j]), sum);
                            }
                            else
                            {
                                if(entry.getKey().toString().compareTo(tokens[j])<0 && (!entry.getKey
                                {
                                  individual_hash.put(new Text(tokens[j]), one);
                                }
                            }
                        }
                        context.write(new Text(entry.getKey().toString()) , individual_hash);
                        individual_hash.clear();
                    }
                }
                all_hash.clear();
            }
```

Reducer of Stripes approach

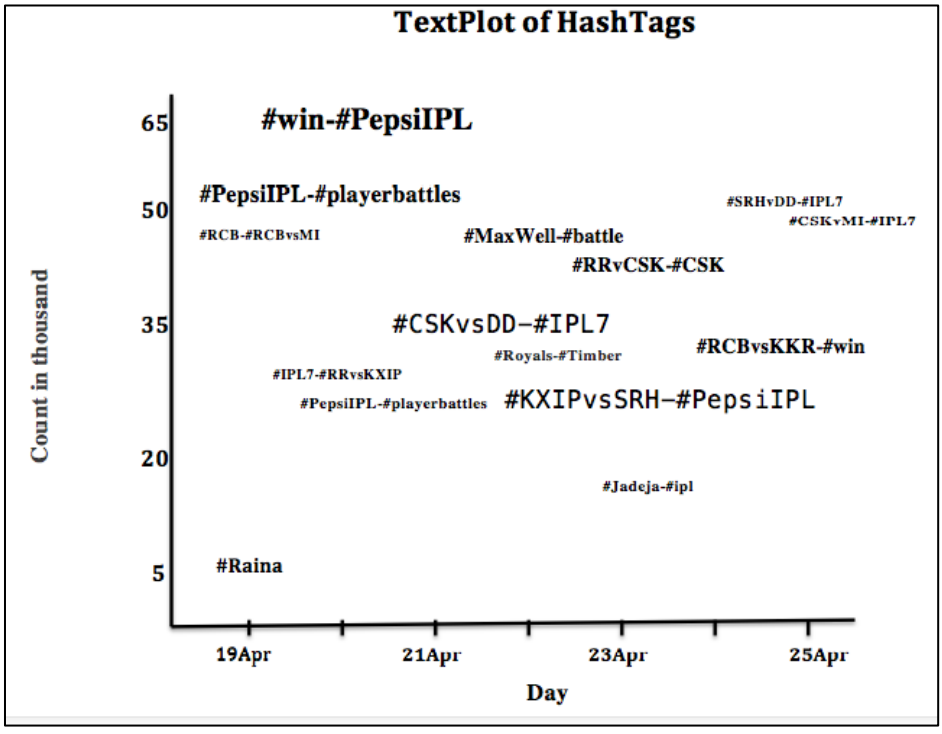
TokenizerMapper.java | IntSumReducer.java

```
extends Reducer<Text,MapWritable,Text,DoubleWritable> {
    private IntWritable result = new IntWritable();

    private final static IntWritable one = new IntWritable(1);
    public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, I
        MapWritable all_hash = new MapWritable();
        for (MapWritable val : values)
        {
            for (Entry<Writable, Writable> entry : val.entrySet())
            {
                if(all_hash.containsKey(entry.getKey()))
                {
                    int finalValue=  Integer.parseInt(all_hash.get(entry.getKey()).toString())+Intege
                    all_hash.put(entry.getKey(), new IntWritable(finalValue));
                }
                else
                {
                    all_hash.put(entry.getKey(), entry.getValue());
                }
            }
        }
        int sum = 0;

        for (Entry<Writable, Writable> entry : all_hash.entrySet())
        {
            sum = sum + Integer.parseInt(entry.getValue().toString());
        }
        for (Entry<Writable, Writable> entry : all_hash.entrySet())
        {
            String finalkey = key.toString()+","+entry.getKey().toString()+",";
             double rf = Double.parseDouble(entry.getValue().toString()) / sum;
            context.write(new Text(finalkey), new DoubleWritable(rf));
        }
```

Following Pie chart displays the top 10 co-occurring hash tag pairs with their counts.



The following plot shows day-wise co-occurring hash tags with highest occurrences.



C. To cluster the tweeters by the number of followers they have based on low, high and average followers count. The actual average value for each cluster will depend on your data size. This information may be used for marketing and for targeting the ads/messages.

Mapreduced version of K-means

Configuration of the cluster used (if one was used)

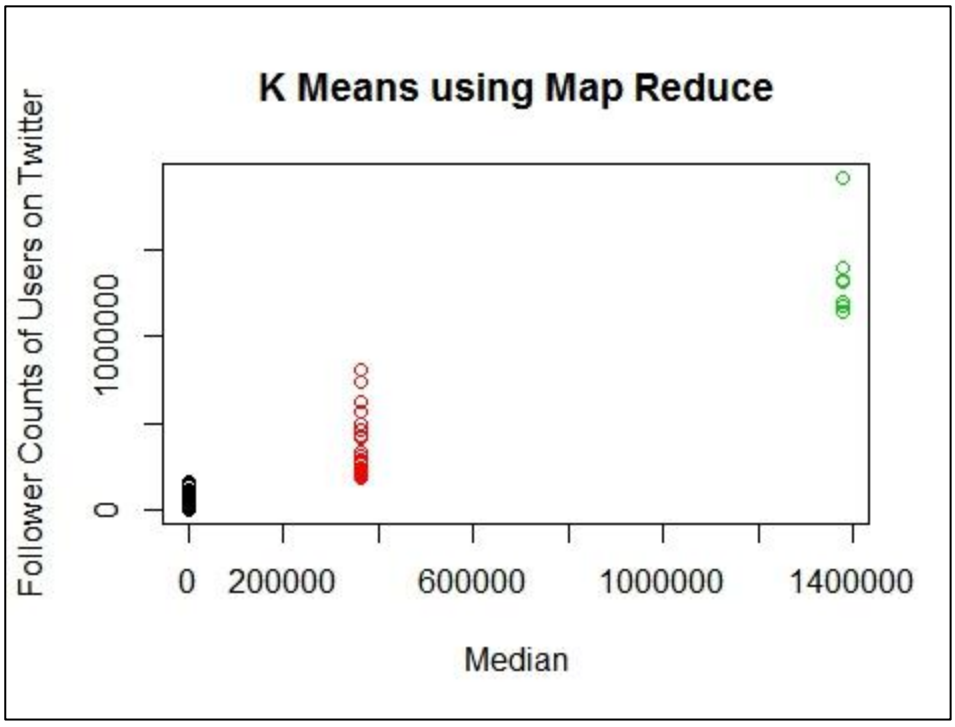
Algorithm:

```
Class Mapper
  method Map(docid a; doc d)

  minDistance <-Double.MAXDISTANCE;
  for (i = 0; i < k;i + +)
  {
        if (distance(point, clusters[i]) < minDistance)
   {
        minDistance = distance(point, clusters[i]);
        group = i;
   }
  }
EmitIntermediate(group, point);

Class Reducer
  method Reduce(group ,Iterator < PointWritable > points)
    {
        num = 0;
        while (points.hasNext())
        {
            PointWritable currentPoint = points.next();
            num+ = currentPoint.get Num();
            for (i = 0;i < dim; i + +)
            sum[i]+ = currentPoint.point[i];
        }
        for (i = 0; i < dimension;i + +)
        mean[i] = sum[i]/num;
    Emit(key, mean);
}
```

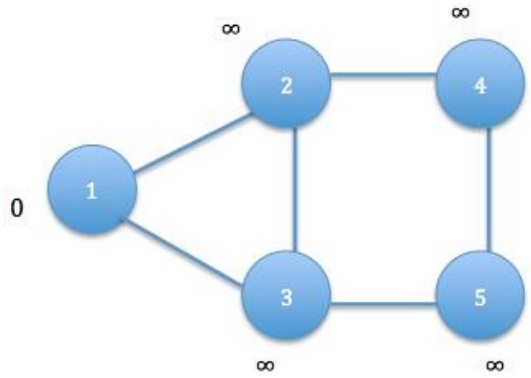
Following is the code snippet of the Mapper of K-means algorithm.

```
tSumReducer.java   TokenizerMapper.java   *TokenizerMapper.java   IntSu

            String str_arr[] = value.toString().split(",");

            String grp_no = str_arr[1];
            String foll_cnt = str_arr[2];
            String iter_rec = str_arr[3];
            String user_name = str_arr[4];
            int f_cnt = Integer.parseInt(foll_cnt.replaceAll(" ", ""));
            IntWritable count = new IntWritable(f_cnt);
            int grp1 = Math.abs( f_cnt - median_1);
            int grp2 = Math.abs(f_cnt - median_2);
            int grp3 = Math.abs(f_cnt - median_3);
            String grp = null ;
            int median = 0;

            if(grp1 <= grp2 && grp1 <= grp3 )
            {
                grp = "grp1";
                median = median_1;
            }
            else if(grp2 <= grp1 && grp2 <= grp3 )
            {
                grp = "grp2";
                median = median_2;
            }
            else if(grp3 <= grp1 && grp3 <= grp2 )
            {
                grp = "grp3";
                median = median_3;
            }
            String send = foll_cnt+","+median+","+iter_rec+","+user_name;
            context.write(new Text (grp), new Text(send));
        }
```

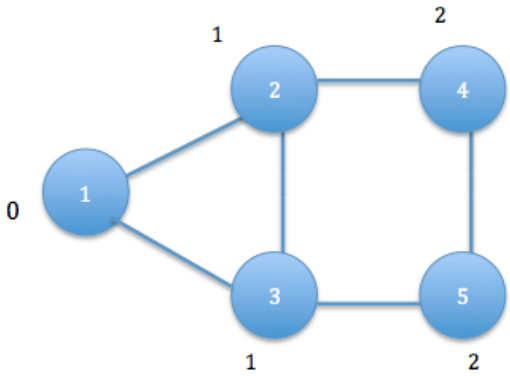
Following plot displays the 3 clusters of users formed depending on their follower counts and the respective medians of the clusters



D. To develop the "All Pair Shortest Path" for given nodes and their adjacency list.



- The above input graph is taken as input. The information regarding the source node and the adjacency nodes is available. The aim is to update the distance of each node from the source node.
- An iterative Map Reduce implementation is used to find the all source shortest path.
- Output of one Map Reduce is recursively given as input and the distances are updated.
- Convergence is obtained when the distance of each node from the source remains unchanged.
- The following graph is obtained as the output with correctly updated distances.



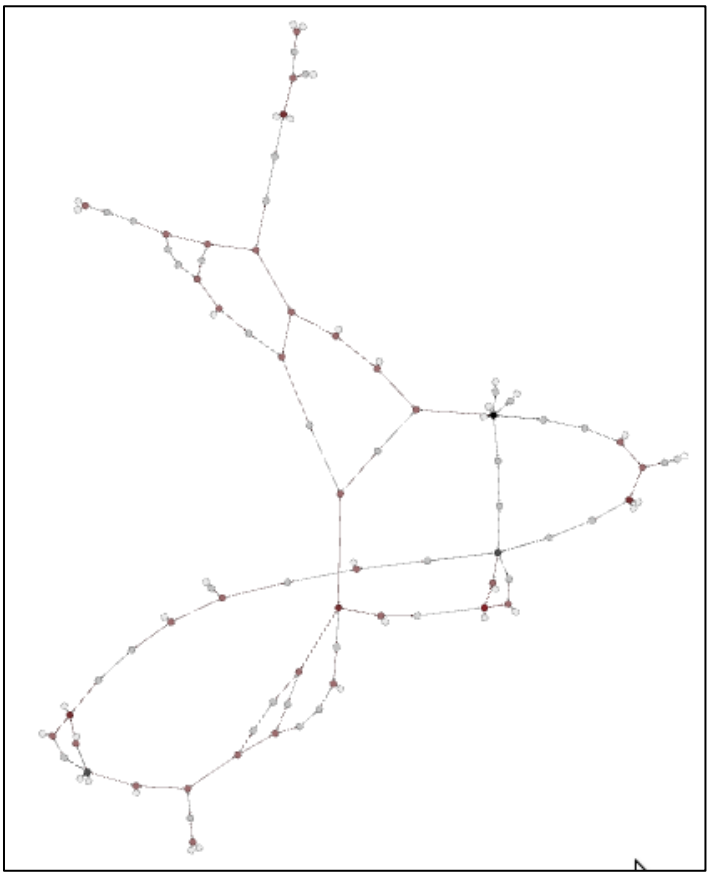
Following is the code Snippet for the recursive implementation of map reduce and the checking of convergence condition.

CustomPartitioner.java | WordCount.java

```
        //recursive implementation of Map Reduce
        while(true)
        {
            HashMap <Integer, Integer> graph_curr = new HashMap<Integer, Integer>();
            Path path = new Path(outputFilePath);
            FileSystem fs = FileSystem.get(new Configuration());
            BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(path)));

            String line = br.readLine();
            while(line!=null)
            {
                String [] all = line.split(" ");
                String node  = all[0];
                String distance  = all[1];
                String adj  = all[2];
                int nd = Integer.parseInt(node.toString());
                int dist = Integer.parseInt(distance.toString());
                graph_curr.put(nd,dist);
                line = br.readLine();
            }
            //Convergence condition checking
            if (graph_prev.isEmpty())
            {
                equal = false;
            }
            else
            {
                for (Entry<Integer, Integer> entry : graph_curr.entrySet())
                {
                    int key_curr = entry.getKey();
                    int value_curr = entry.getValue();


                    if (graph_prev.containsKey(key_curr))
                    {
                        int val_prev = graph_prev.get(key_curr);
                        if(value_curr == val_prev)
                        {
                            equal = true;
                        }
                        else
                        {
                            equal = false;
                            break;
                        }
                    }
                    else
                    {
                        equal = false;
                        break;
                    }
                }
            }


            if (equal)
            {
                break;
            }

            graph_prev.clear();
            graph_prev.putAll(graph_curr);
            job = Job.getInstance(conf);
            job.setJarByClass(WordCount.class);
            job.setMapperClass(TokenizerMapper.class);
            //job.setCombinerClass(IntSumReducer.class);
            job.setReducerClass(IntSumReducer.class);
            // Set the output Key type for the Mapper
            job.setMapOutputKeyClass(Text.class);
            // Set the output Value type for the Mapper
            job.setMapOutputValueClass(Text.class);
            // Set the output Key type for the Reducer
            job.setOutputKeyClass(Text.class);
            // Set the output Value type for the Reducer
            job.setOutputValueClass(Text.class);
            inputPath = outputFilePath;
            outputPath = _output + w;
            outputFilePath=outputPath + outputFile;

            FileInputFormat.addInputPath(job, new Path(inputPath));
            FileOutputFormat.setOutputPath(job, new Path(outputPath));

            success=job.waitForCompletion(true) ? 0 : 1;
            w++;
        }
        return success;
    }
```

Output of Large graph:

The following image is generated using Gephi by providing the output of running map reduce on the given input large graph.



7. Lessons learned

This project gave us hands on experience of handling and parallel processing of huge amount of data. Data collection process introduced us to java twitter streaming API. It was very interesting to gather and then aggregate the social networking data so as to extract interesting patterns and recent trends from it. We got exposure to work with prominent parallel data processing tool: Hadoop2. Apache Hadoop2 framework is gaining significant momentum from both industry and academia as the volume of data to analyze grow rapidly. This project helped us not only to gain knowledge about installation and configuration of hadoop distributed file system but also map reduce programming model. At the end of analysis phase data visualization was performed in RStudio.