

# **CSE 587- Data Intensive Computing Project II**

## **Big Data Content Retrieval , Storage and Analysis**

Anand Nandan Menon

UB # : 50098027

Email: [anandnan@buffalo.edu](mailto:anandnan@buffalo.edu)

Subhranil Banerjee

UB # : 50096800

Email: [subhrani@buffalo.edu](mailto:subhrani@buffalo.edu)

## **Abstract:**

Big data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, curation, storage, search, sharing, transfer, analysis and visualization. The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions."

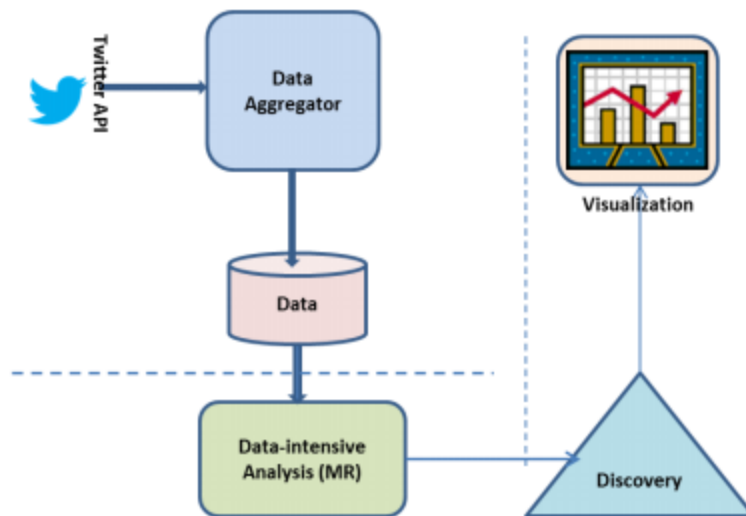
In this project, we shift our focus to one of the most popular websites of the world, Twitter, a gigantic hub of data accessed and modified , every second by millions of users all over the world. To analyze this data, we utilize the power of parallel processing given to us by the Mapreduce programming model.

## **Objective:**

To analyze Twitter data using the Mapreduce programming model and using various visualization tools on the output .

## Project Approach:

Our approach can be summarized in three major steps. The following figure further enunciates them.



- Collecting data from twitter's live feed using the Streaming API's provided.
- Feeding this data to a Hadoop 2.20 cluster for analysis.
- Visualizing the output using various tools like R and Gephi.

## **Data Aggregation:**

For the purpose of this project , we used the *TwitterStream* Api's provided by twitter to get data off the public feed.

Aggregator: Our custom application *TweetAggregator* uses the above mentioned api's by way of a public and secret key pair . It implements a listener class for receiving notifications whenever a tweet is captured. For every tweet, the following information is maintained.

- Text of the tweet.
- Screen name of user who tweeted.
- Follower Count of the user.

This data is stored locally on files. Each such file contain 10000 tweets. The number 10000 was chosen as a tradeoff between the size of an individual file (to avoid exceeding the JVM's heap for in-mapper combining) and the number of mappers.

Cleaning: Our custom application *TweetCleaner* takes each of the above mentioned files as input and for every line (tweet) therein, removes punctuations , stop words and extra spaces . The output of this application is used as input for our mapreduce jobs.

Volume of Data Collected: We collected a total of 1.75 million tweets in two sets of 1 million and 750k respectively. However the results from the first set were discarded as they were deemed “uninteresting”.

## **Goals:**

### **1.WordCount:**

**Requirement Specification:** To find out the most trending words, hashtags and mentions.

**Implementation Details:** The algorithm comprises two parts:

- The first phase implements a similar logic for wordcount as given in chapter 4 of Lin and Dyer's TextBook. We have built our custom partitioner class *WordCountPartitioner* to streamline the output of mapper into 3 reducers , one each for words, hashtags and mentions.

Mapper: TokenizerMapper

Combiner: In-mapper

Partitioner: WordCountPartitioner

Reducer: IntSumReducer

PseudoCode:

```
class Mapper
  method Initialize
    H<- new AssociativeArray
  method Map(docid a, doc d)
    for all term t ∈ doc d do
      H{t} ← H{t} + 1
  method Close
    for all term t ∈ H
      Emit(term t, count H{t})
```

```

class Reducer
  method Reduce(term t, counts [c 1 , c 2 , . . .])
    sum <- 0
    for all count c ∈ counts [c 1 , c 2 , . . .] do
      sum <-sum + c
    Emit(term t, count sum)

```

- The second phase involves secondary sorting using a combined key(word + count). Our partitioner uses the *compareTo()* method of the custom *PairKey* class to sort it into descending order by count.

```

Mapper : SorterMapper
Partitioner : SorterPartitioner
Reducer : SorterReducer

```

PseudoCode:

```

class Mapper
  method Map(docid a, doc d)
    for all term t ∈ doc d do
      emit(t.string+t.count,t.count)

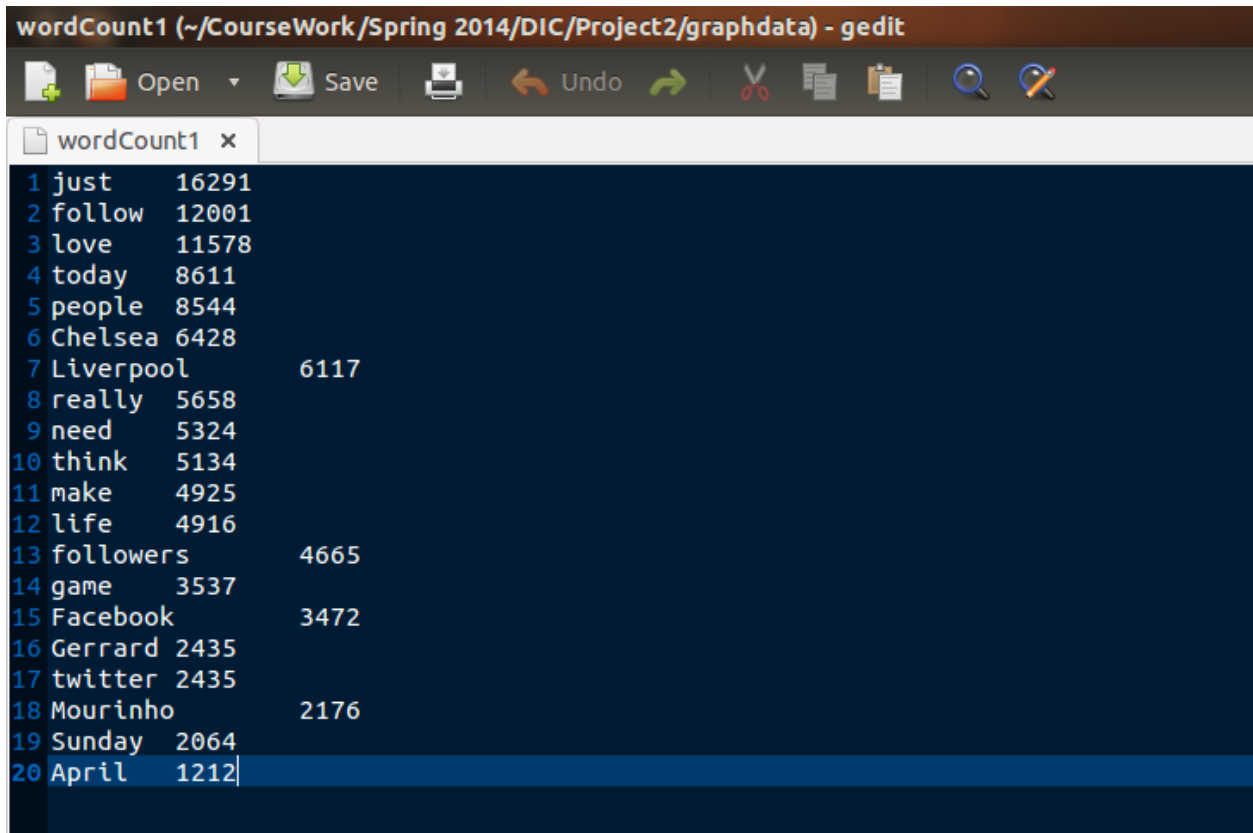
```

```

class Reducer
  method Reduce(term t, counts [c1 ,c2 , . . .])
    sum <- 0
    for all count c ∈ counts [c1,c2 , . . .] do
      sum <-sum + c
    Emit(term t, count sum)

```

**Results :**After running the Mapreduce tasks for wordcount on the aggregated data , the following were the top trending results.



The image shows a screenshot of a gedit text editor window. The title bar reads "wordCount1 (~/.CourseWork/Spring 2014/DIC/Project2/graphdata) - gedit". The editor contains a list of 20 words and their corresponding counts, numbered 1 through 20. The words are listed in a single column, and the counts are listed in a second column. The words are: just, follow, love, today, people, Chelsea, Liverpool, really, need, think, make, life, followers, game, Facebook, Gerrard, twitter, Mourinho, Sunday, and April. The counts are: 16291, 12001, 11578, 8611, 8544, 6428, 6117, 5658, 5324, 5134, 4925, 4916, 4665, 3537, 3472, 2435, 2435, 2176, 2064, and 1212. The word "April" is highlighted in blue.

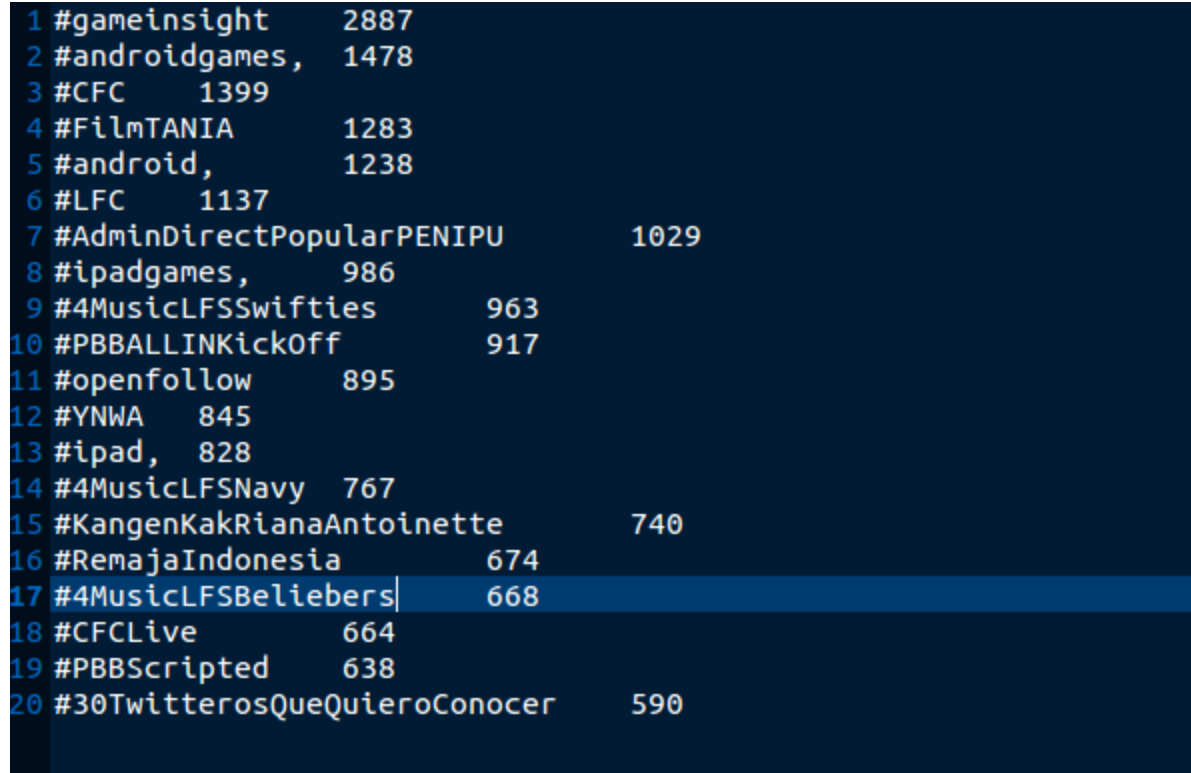
Rank	Word	Count
1	just	16291
2	follow	12001
3	love	11578
4	today	8611
5	people	8544
6	Chelsea	6428
7	Liverpool	6117
8	really	5658
9	need	5324
10	think	5134
11	make	4925
12	life	4916
13	followers	4665
14	game	3537
15	Facebook	3472
16	Gerrard	2435
17	twitter	2435
18	Mourinho	2176
19	Sunday	2064
20	April	1212

Fig 2. Counts for the top 20 trending words



Fig 3. Tag graph representation for the top 20 trending words





1	#gameinsight	2887	
2	#androidgames,	1478	
3	#CFC	1399	
4	#FilmTANIA	1283	
5	#android,	1238	
6	#LFC	1137	
7	#AdminDirectPopularPENIPU		1029
8	#ipadgames,	986	
9	#4MusicLFSSwifties	963	
10	#PBBALLINKickOff	917	
11	#openfollow	895	
12	#YNWA	845	
13	#ipad,	828	
14	#4MusicLFSNavy	767	
15	#KangenKakRianaAntoinette		740
16	#RemajaIndonesia	674	
17	#4MusicLFSBeliebers	668	
18	#CFCLive	664	
19	#PBBScripted	638	
20	#30TwitterosQueQuieroConocer		590

Fig 4. Counts for the top 20 trending hashtags

#kangenkakrianaantoinette  
#4musiclfsbeliebers #4musiclfsnavy  
#remajaindonesia#pbballinkickoff  
#admindirectpopularpenipu  
#cfclive #filmmtania  
#pbbscripted #lfc #cfc #ynwa  
**#gameinsight**  
#androidgames,  
#android, #ipad,  
#ipadgames,  
#4musiclfsswifties  
#openfollow  
#30twitterosquequieroconocer

Fig 3. Tag graph representation for the top 20 trending hashtags

@BieberAnnual:	3122
@justinbieber:	2482
@NiallOfficial	2359
@YouTube	1639
@chelseafc:	1036
@Gabriele_Corno:	935
@louis_tomlinson	912
@Harry_Styles:	556
@onedirection	550
@LFC:	419

Fig 5. Counts for the top 10 trending mentions

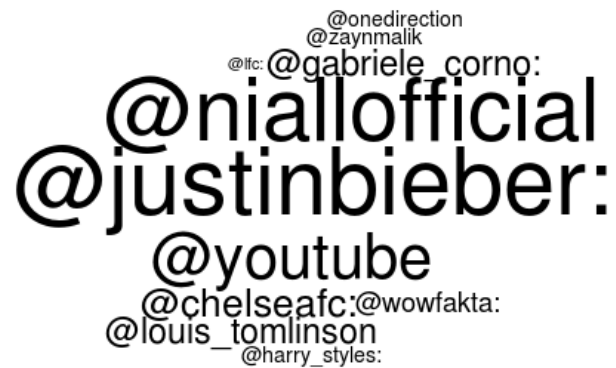


Fig 6. Tag graph representation for the top 10 trending mentions

## **2. Word Co-occurrence**

### **Requirement Specification:**

The task at hand is to find out the co-occurring words in all the tweets that we have collected thus far. The tweets collected were cleaned to remove stop words and unnecessary punctuations to arrive at a clean data set after the required Exploratory data analysis.

When co-occurrence is considered we mainly have two approaches. We can consider all possible pair of words in a single tweet, or we can keep track of occurrences using an associative array. The former approach is rightfully named as Pairs and the latter as Stripes.

The requirement is to generate the relative co-occurrence of words, in both of these approaches.

### **Implementation Details:**

For Stripes approach we scan each word and emit a ([word,neighbour],count) key value pair, one for each of its neighbours in a single tweet. Since this would only give us the regular count we are also emitting a special key pair as discussed in the Text illustrated as ([Word,\*],count). This special pair would give us all possible counts of occurrences of 'Word'. The emitted pairs are then partitioned only on the basis of first key so that all word pairs including special pair arrive at the same reducer. To ensure that the special pair reaches before regular pair, we make use of compareTo function in the WritableComparable class. Hence at the reducer we can just proceed to divide the regular counts by the marginal or total counts to arrive at the relative frequency for each pair.

Mapper:PairsMapper  
Partitioner:PairsPartitioner  
Reducer:PairReducer  
Driver:PairsDriver

In the case of Stripes, this approach becomes all the more easier. This is because since we are obtaining the result in an associative array or specifically a MapWritable, we can easily calculate the total as well as relative occurrence at the reducer.

Mapper:StripesMapper  
Reducer:StripesReducer  
Driver:StripesDriver

PseudoCode

```
Class Mapper(Object, Text)
{
  for each line in Text
  Output special pair
  for each neighbour
  output regular pair,count
}
```

```
Class Reducer(PairsKey, IntWritable)
{
  for each input key pair
  If special then set total count
  else
  if regular pair then set count/total to get relative count
  output the same
}
```

## Stripes

Class Mapper(Object, Text)

{

for each line

for each word add neighbours into associative array

output key and MapWritable containing neighbours and counts

}

Class Reducer(Text, MapWritable)

{

For each map add the corresponding entries element wise.

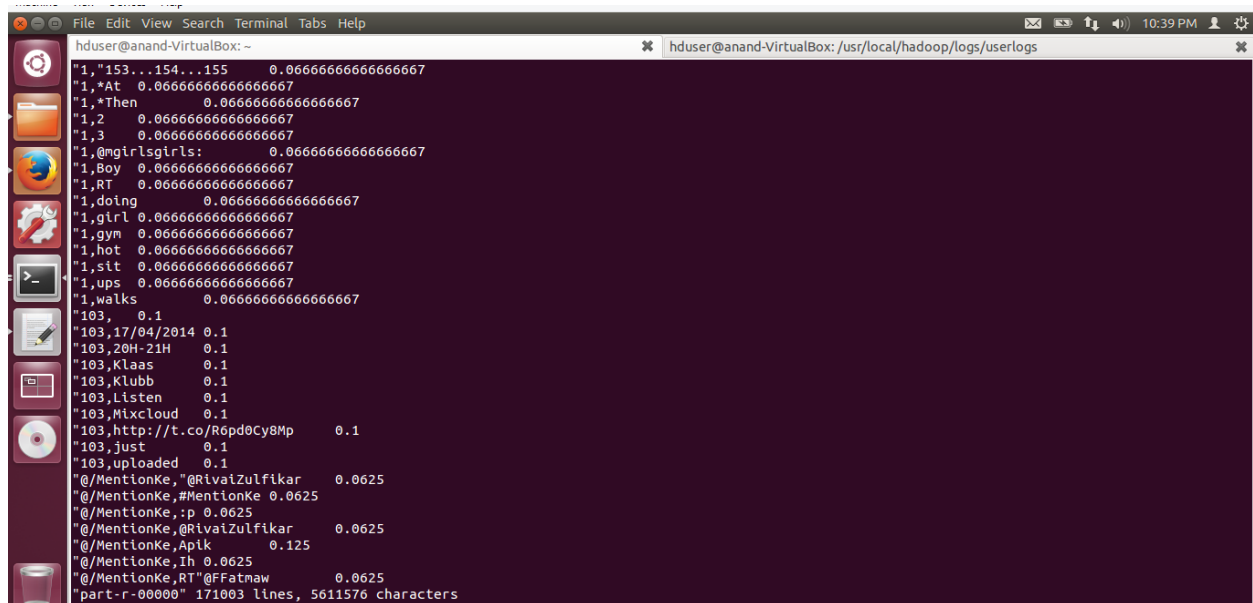
For each word in Text get count from all of the entries in Map to get total

Emit key, and each neighbour with count/total count to get relative count.

}

## Results:

For the results we have the following screen shots indicating the reducer outputs. We have tested our implementation with 3 reducers and hence have three output files as shown:



```
hduser@anand-VirtualBox: ~
1,153...154...155 0.06666666666666667
1,*At 0.06666666666666667
1,*Then 0.06666666666666667
1,2 0.06666666666666667
1,3 0.06666666666666667
1,@girlsgirls: 0.06666666666666667
1,Boy 0.06666666666666667
1,RT 0.06666666666666667
1,doing 0.06666666666666667
1,girl 0.06666666666666667
1,gym 0.06666666666666667
1,hot 0.06666666666666667
1,sit 0.06666666666666667
1,ups 0.06666666666666667
1,walks 0.06666666666666667
103, 0.1
103,17/04/2014 0.1
103,20H-21H 0.1
103,Klaas 0.1
103,Klubb 0.1
103,Listen 0.1
103,Mixcloud 0.1
103,http://t.co/R6pd0Cy8Mp 0.1
103,just 0.1
103,uploaded 0.1
@/MentionKe,@Rivaizulfikar 0.0625
@/MentionKe,#MentionKe 0.0625
@/MentionKe,:p 0.0625
@/MentionKe,@Rivaizulfikar 0.0625
@/MentionKe,Apik 0.125
@/MentionKe,Ih 0.0625
@/MentionKe,RT@FFatmaw 0.0625
part-r-000000 171003 lines, 5611576 characters
```

Fig 7: Pairs output: part -r- 000000

```
hduser@anand-VirtualBox: ~  
hduser@anand-VirtualBox: ~  
Dash Home      udah      0.14285714285714285  
@AmeeraAmzah,@FahmiQuotes:      0.1  
@AmeeraAmzah,RT      0.1  
@AmeeraAmzah,Renungan      0.1  
@AmeeraAmzah,dan      0.1  
@AmeeraAmzah,dari      0.1  
@AmeeraAmzah,diri      0.1  
@AmeeraAmzah,http://t.co/BQn0RcFlny      0.1  
@AmeeraAmzah,kita      0.1  
@AmeeraAmzah,muhasabah      0.1  
@AmeeraAmzah,sekarang      0.1  
@AyuArska,@rentherwant      0.14285714285714285  
@AyuArska,blm      0.14285714285714285  
@AyuArska,disana      0.14285714285714285  
@AyuArska,lo      0.14285714285714285  
@AyuArska,mutuun      0.14285714285714285  
@AyuArska,otw      0.14285714285714285  
@AyuArska,udh      0.14285714285714285  
@BedaFamiglia,#LRHondaGancy"      0.3333333333333333  
@BedaFamiglia,Flying      0.3333333333333333  
@BedaFamiglia,JKT48      0.3333333333333333  
@CocoNanya,Ayam      0.125  
@CocoNanya,apa      0.125  
@CocoNanya,barang      0.125  
@CocoNanya,beli      0.125  
@CocoNanya,ingin      0.125  
@CocoNanya,kamu      0.125  
@CocoNanya,penyet      0.125  
@CocoNanya,yang      0.125  
@DesssKT,@KingMoz4      0.06666666666666667  
@DesssKT,@KingMoz4:      0.06666666666666667  
@DesssKT,I'll      0.06666666666666667
```

Fig 8: Pairs output:part -r- 00001

```
hduser@anand-VirtualBox: ~  
hduser@anand-VirtualBox: ~  
hmm,day      0.125  
hmm,dorang      0.125  
hmm,family      0.125  
hmm,jeles      0.125  
hmm,nye      0.125  
hmm,pgl      0.125  
hmm,seronok      0.125  
hmm,siot      0.125  
#ZaynMalikCestUnAlgerien,ange      0.1  
#ZaynMalikCestUnAlgerien,au      0.1  
#ZaynMalikCestUnAlgerien,branle      0.1  
#ZaynMalikCestUnAlgerien,c'est      0.2  
#ZaynMalikCestUnAlgerien,ce      0.1  
#ZaynMalikCestUnAlgerien,mec      0.1  
#ZaynMalikCestUnAlgerien,pire      0.1  
#ZaynMalikCestUnAlgerien,s'en      0.1  
#ZaynMalikCestUnAlgerien,tout      0.1  
..dalam,(HR      0.07692307692307693  
..dalam,@DoaIndah:      0.07692307692307693  
..dalam,Muslim      0.07692307692307693  
..dalam,RT      0.07692307692307693  
..dalam,dosa      0.07692307692307693  
..dalam,menimpa      0.07692307692307693  
..dalam,musibah      0.07692307692307693  
..dalam,muslim      0.07692307692307693  
..dalam,penghapusan      0.07692307692307693  
..dalam,seorang      0.07692307692307693  
..dalam,setiap      0.07692307692307693  
..dalam,terdapat      0.07692307692307693  
..dalam,yang      0.07692307692307693  
@Adhel_nabartba,"melanhia_elha      0.0625  
@Adhel_nabartba,"Adhel_nabartba      0.0625  
part-r-00002" 186500 lines, 6134663 characters
```

Fig 9 : Pairs output : part-r-00002



```
File Edit View Search Terminal Tabs Help
hduser@anand-VirtualBox: ~
hduser@anand-VirtualBox: /usr/local/hadoop/logs/userlogs

1,*Then 0.06666666666666667
1,"153...154...155 0.06666666666666667
1,doing 0.06666666666666667
1,Boy 0.06666666666666667
1,girl 0.06666666666666667
1,sit 0.06666666666666667
1,@girlsgirls: 0.06666666666666667
1,2 0.06666666666666667
1,hot 0.06666666666666667
1,gym 0.06666666666666667
1,3 0.06666666666666667
1,walks 0.06666666666666667
1,ups 0.06666666666666667
1,*At 0.06666666666666667
1,RT 0.06666666666666667
103, 0.1
103,Klaas 0.1
103,Listen 0.1
103,Klubb 0.1
103,uploaded 0.1
103,20H-21H 0.1
103,http://t.co/R6pd0Cy8Mp 0.1
103,Mixcloud 0.1
103,17/04/2014 0.1
103,just 0.1
@/MentionKe,yg 0.07142857142857142
@/MentionKe,opene 0.07142857142857142
@/MentionKe,th 0.07142857142857142
@/MentionKe,@Rtvaizulfikar 0.07142857142857142
@/MentionKe,th 0.07142857142857142
@/MentionKe,RT@FFatmaw 0.07142857142857142
@/MentionKe,:p 0.07142857142857142
```

Fig 10 : Stripes output: part-r-00000

```
hduser@anand-VirtualBox: ~
hduser@anand-VirtualBox: /usr/local/hadoop/logs/userlogs

@AneeraAmzah,sekarang 0.1
@AyuArska,disana 0.14285714285714285
@AyuArska,mutuun 0.14285714285714285
@AyuArska,@renitherwani 0.14285714285714285
@AyuArska,otw 0.14285714285714285
@AyuArska,blm 0.14285714285714285
@AyuArska,lo 0.14285714285714285
@AyuArska,udh 0.14285714285714285
@BedaFamiglia,Flying 0.3333333333333333
@BedaFamiglia,#LRHondaGancy 0.3333333333333333
@BedaFamiglia,JKT48 0.3333333333333333
@CocoNanya,ingin 0.125
@CocoNanya,yang 0.125
@CocoNanya,apa 0.125
@CocoNanya,kamu 0.125
@CocoNanya,beli 0.125
@CocoNanya,Ayam 0.125
@CocoNanya,barang 0.125
@CocoNanya,penyet 0.125
@DesssKT,U 0.07142857142857142
@DesssKT,b 0.07142857142857142
@DesssKT,right 0.07142857142857142
@DesssKT,know 0.07142857142857142
@DesssKT,I'll 0.07142857142857142
@DesssKT,Lol 0.07142857142857142
@DesssKT,can't 0.07142857142857142
@DesssKT,like 0.07142857142857142
@DesssKT,RT 0.07142857142857142
@DesssKT,@KingMoz4 0.07142857142857142
@DesssKT,ok 0.07142857142857142
@DesssKT,nice 0.07142857142857142
@DesssKT,stmps 0.07142857142857142
```

Fig 11 : Stripes output: part-r-00001



Partitioner: None (Single Reducer)

PseudoCode:

class Mapper

method Initialize()

CentroidList=readFromFile(“CentroidsPath”)

method Map(docid a, doc d)

for all term  $t \in \text{doc } d$  do

minDistance=Inf.

for( $c : \text{CentroidList}$ )

distance=abs( $t.\text{followerCount}-c$ )

if( $\text{distance} < \text{minDistance}$ )

minDistance=distance

nearestCentroid= $c$

emit(nearestCentroid, $t$ )

class Reducer

method Initialize()

Counter=0

newCentroids=new List()

method Reduce(Centroid  $c$ , Terms[ $t_1, t_2, \dots$ ])

sum=0

count=0

for( $t$ : Terms)

sum=sum+ $t.\text{followerCount}$

count=count+1;

emit( $c, t$ )

avg=sum/count

if( $\text{avg} \neq c$ )

Counter=Counter+1

```
newCentroids.add(avg)
```

```
method Close()
```

```
newCentroids.writeToFile("CentroidsPath")
```

**Result:** We took our initial centroids at 100, 2000 and 10000 and the final centroids for the clusters (after 21 iterations) were 1522, 58697 and 3645047

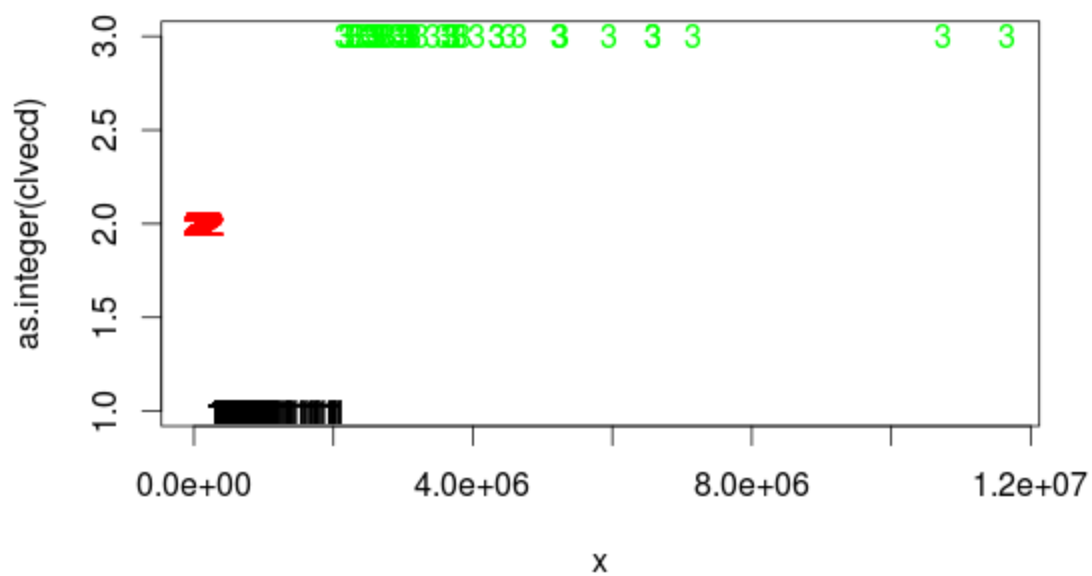


Fig 13 : Clusters plotted using R

## **4. Shortest Path in Graph**

The task at hand is to implement the MapReduce version of the Dijkstra's Algorithm for shortest paths to all nodes of the given graph from a single source node. Here we have a connected directed graph as input with adjacency list indicating neighbours and distance stored alongside. We map over all nodes. Mappers emit a key-value pair for each neighbor on the node's adjacency list. Where the key value is the node id of the neighbor and value is current distance to node +1. The logic behind this is if we can reach node n with d distance then all connected nodes of n can be reached with d+1. After shuffle reducers have keys corresponding to nodes and distances about all path leading to the node. The reducer then selects shortest and updates the node.

```
class Mapper
{
map(nid, node N)
d=N.distance
emit(nid ,N)
for all node m adjacent to N
emit(m,d+1)
}
```

```
class reducer
{
reduce(nid m,[d1,d2..])
dmin=infinity
M=null;
for all d in d1,d2..
if d is a node then
M=d
```

```
else if d<dmin
dmin=d
m.distance = dmin
emit(m,M)
}
```

### **Results:**

```
1 1 0 2:
2 2 1 3:
3 3 2 2:13:
4 4 5 14:
5 5 5 14:
6 6 10 7:
7 7 9 6:8:17:
8 8 10 7:20:
9 9 13 10:
10 10 12 9:20:
11 11 5 12:
12 12 4 11:13:22:
13 13 3 3:12:14:
14 14 4 4:5:13:15:
15 15 5 14:16:
16 16 6 15:27:
17 17 8 7:27:
18 18 13 19:35:
19 19 12 18:20:21:
20 20 11 8:10:19:
21 21 13 19:
22 22 5 12:24:
23 23 8 32:
24 24 6 22:32:
25 25 8 26:32:
26 26 8 25:27:
27 27 7 16:17:26:28:33:
28 28 8 27:44:
29 29 17 37:38:
30 30 16 45:
31 31 8 32:
32 32 7 23:24:25:31:40:41:46:
33 33 8 27:42:43:
34 34 17 49:
35 35 14 18:50:
36 36 16 50:
```

Fig 15 : Subset of the output of the output of the Dijkstra's Algorithm

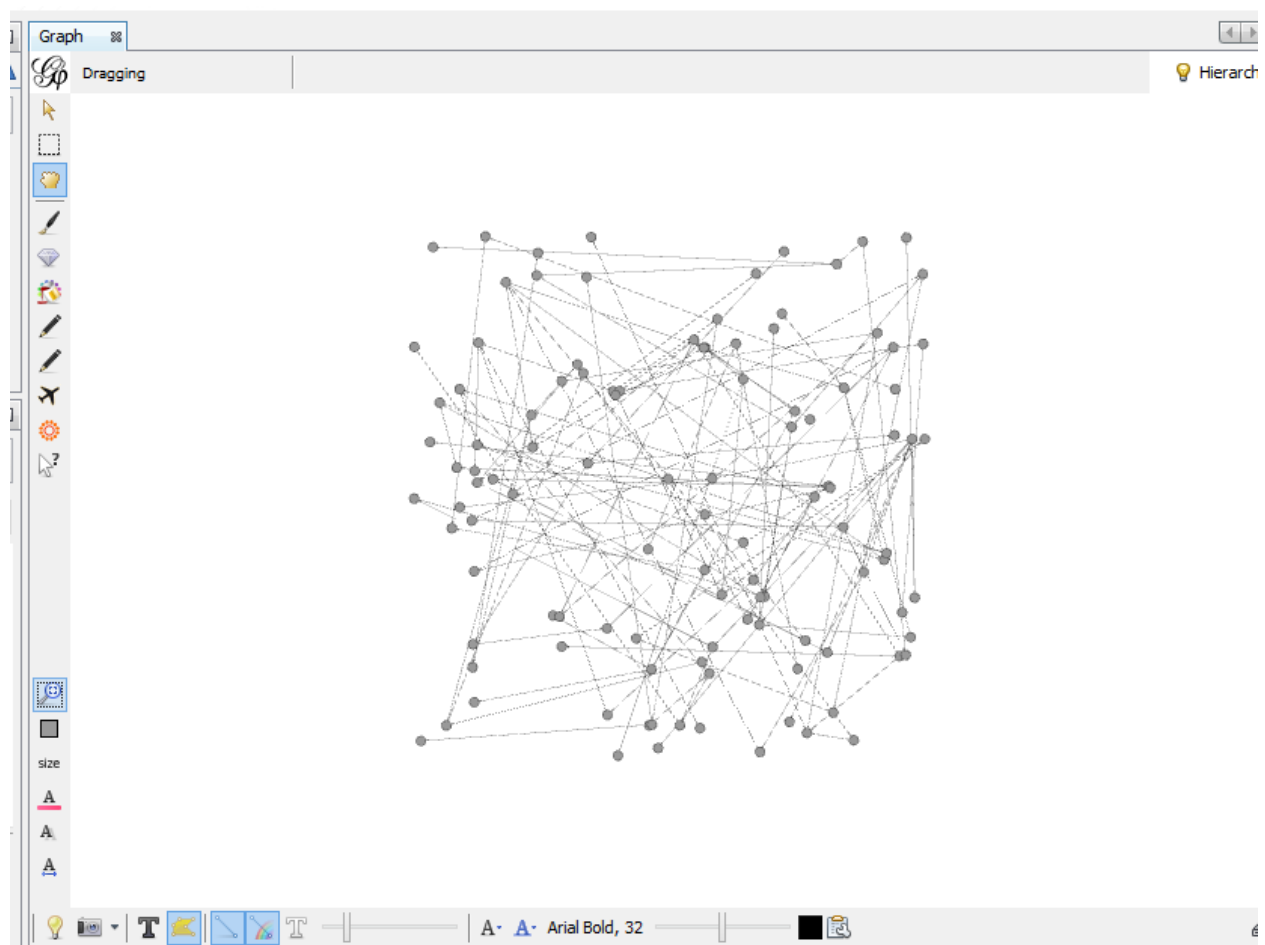


Fig 16 : Visual representation of the given graph (using Gephi)