CSE 587: Data Intensive Computing Project 2

Big-data content retrieval, storage and analysis foundations of data-intensive computing

Project Members:

Gaurav Gosavi (5009 6832)

Ameya Patil (5009 7850)

Contact: gauravgo@buffalo.edu; ameyapat@buffalo.edu

Introduction and brief description

In this project we present analysis done on an aggregated set of 320,443 random tweets collected via the twitter4j streaming API.

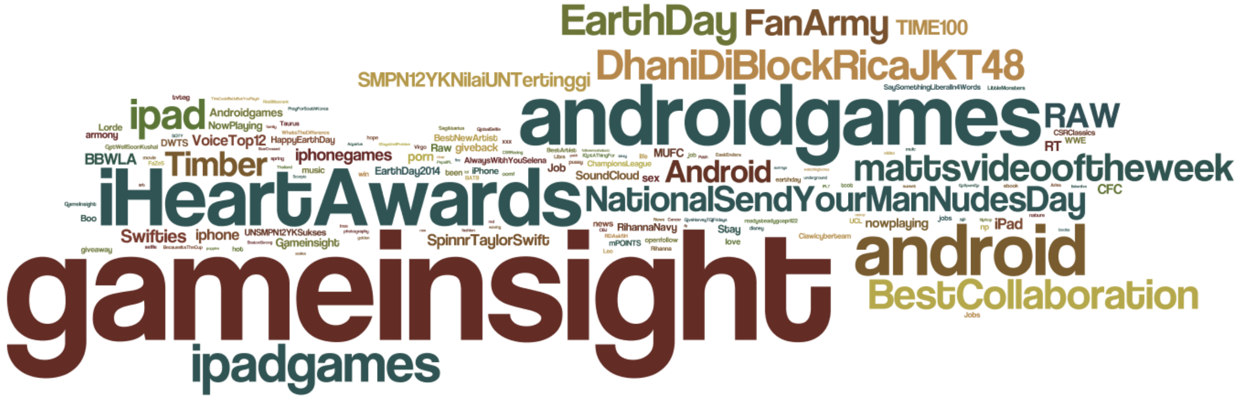
For this data set we used Hadoop to perform tasks like counting the number of Hashtags and @ mentions per tweet, using the MapReduce programming model.

We have also used MapReduce to find out the most trending pair of words and clustering the users according to their number of followers. We have used k means for this.

Also, we have calculated the Shortest Path between two nodes in a given data set using Djikstra's shortest path algorithm using Hadoop and Map Reduce.

Word count.

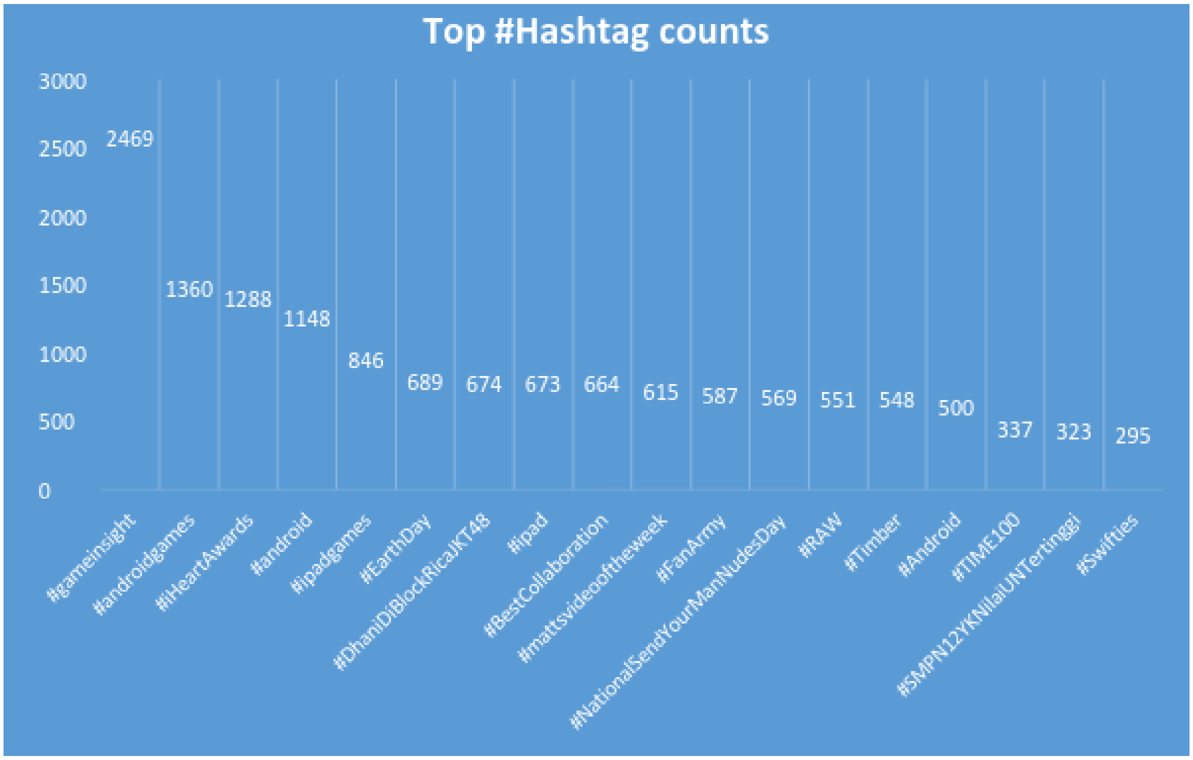
1. Using Map reduce and Hadoop we have implemented the classic word count problem on our aggregated tweet set. The implementation and code can be found in the tar file.



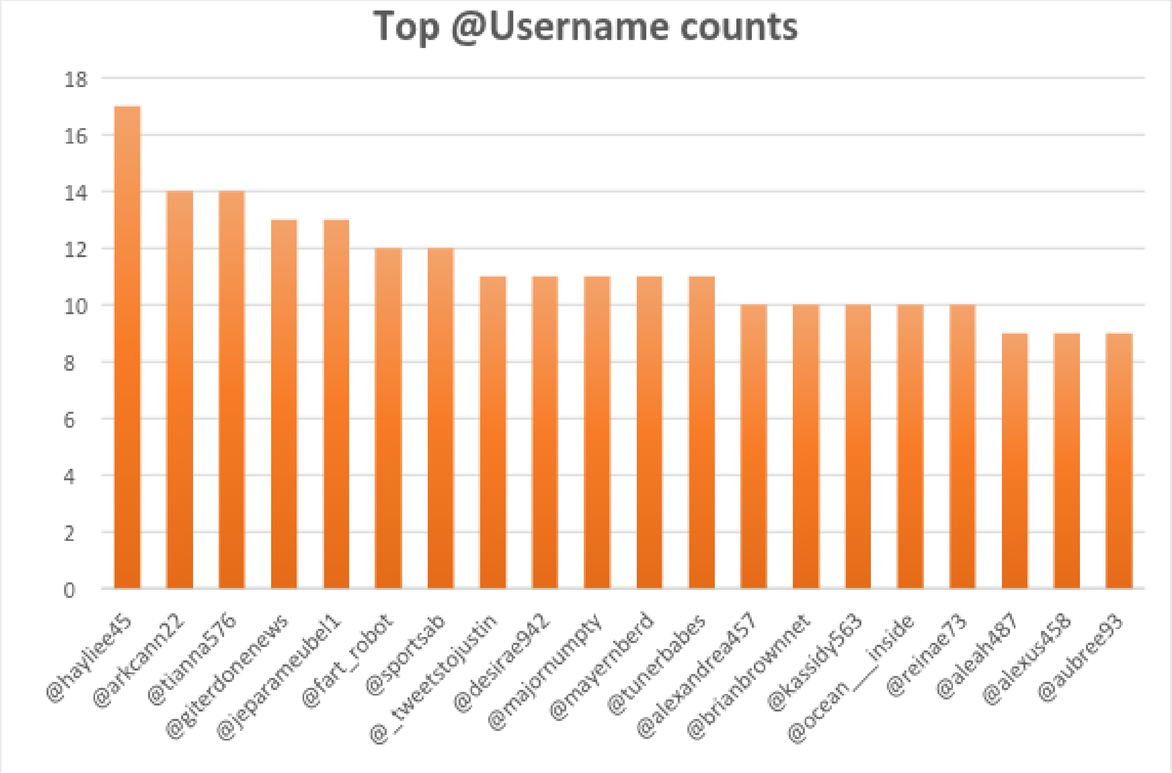
Word cloud of the top trending words.

2. By just tweaking the word count implementation we have found the top trending # word and top trending user (@).

Our Top trending #tags and their respective number of occurrences are:



3. Our top trending @user mentions and their count is:



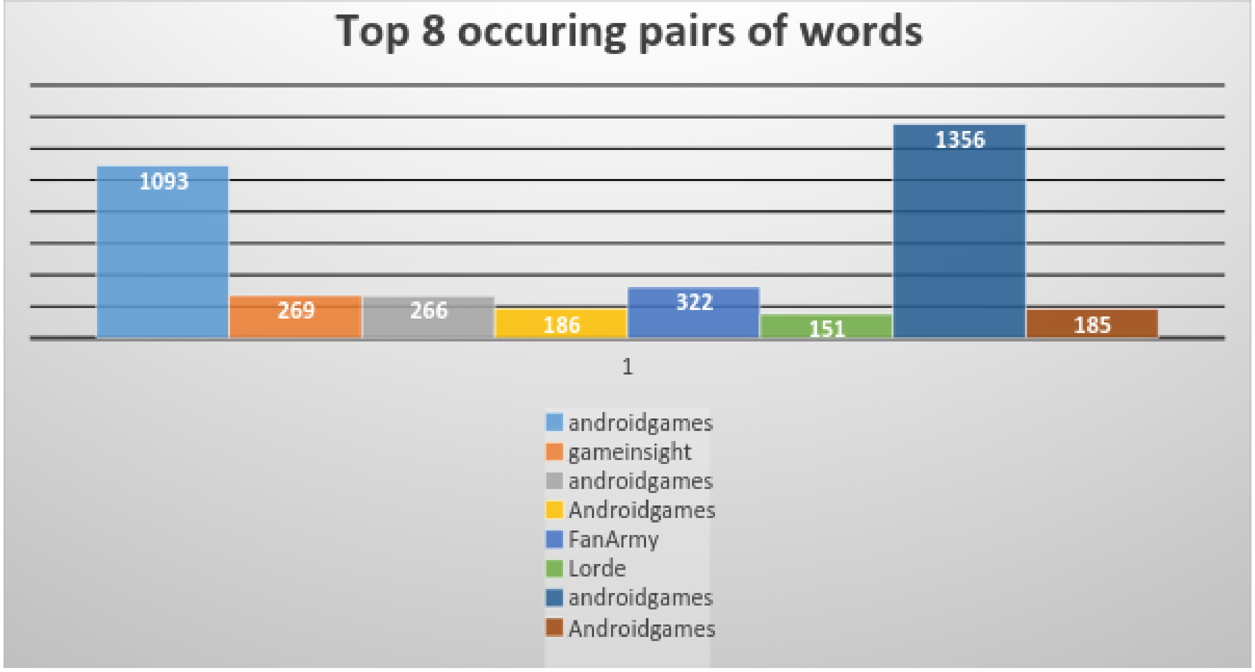
The other @s and #tags and their counts can be found in the 2 CSV files named “hashtagsCount.csv” and “userCounts.csv”.

Co-occurrence

In this section we have calculated the top trending pair of words using the pairs and strips approach as discussed in the Lin and Dyer MR text.

Detailed implementation can be found inside the tar file and its results and the corresponding relative frequency of each pair can be found in the two CSV files named “paircount.csv” and “Stripscount.csv”.

Our top trending pairs of hash tags are:



The algorithms used for strips and pairs approach of finding out the shortest path are as follows:

Pairs:

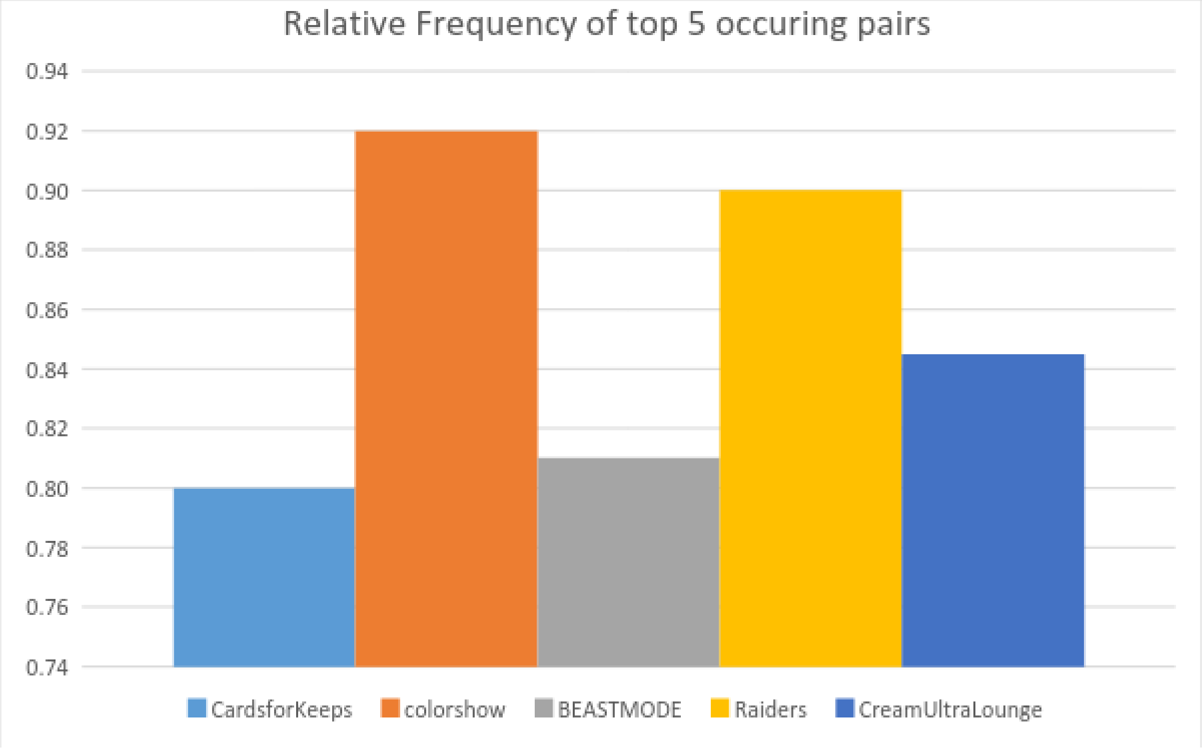
```
class Mapper
method Map(docid a; doc d)
for all term w 2 doc d do
for all term u 2 Neighbors(w) do
Emit(pair (w; u); count 1) . Emit count for each co-occurrence
class Reducer
method Reduce(pair p; counts [c1; c2; : : :])
s   0
for all count c 2 counts [c1; c2; : : :] do
s   s + c . Sum co-occurrence counts
Emit(pair p; count s)
```

Strips:

```
class Mapper
    method Map(docid a; doc d)
        for all term w 2 doc d do
            H   new AssociativeArray
            for all term u 2 Neighbors(w) do
                Hfug   Hfug + 1 . Tally words co-occurring with w
            Emit(Term w; Stripe H)
```

```
class Reducer
    method Reduce(term w; stripes [$H_1$;$H_2$;$H_3$; : : :])
        $H_f$   new AssociativeArray
        for all stripe H 2 stripes [$H_1$;$H_2$;$H_3$; : : :] do
            Sum($H_f$ ;H) . Element-wise sum
        Emit(term w; stripe $H_f$ )
```

Using the above pseudo code we have implemented a Hadoop map reduce program in java to accomplish our task. The detailed code and the .jar file can be found in the tar file.



Single-source Shortest Path

In this project we have also calculated the single-source shortest path between two nodes in a given graph using Hadoop MapReduce.

The algorithm followed is almost the same as Djikstra’s shortest path algorithm.

The pseudo-code for the following is as follows:

```
Dijkstra(G;w; s)
    d[s]   0
    for all vertex v 2 V do
        d[v]   1
    Q   fV g
    while Q 6= ; do
        u   ExtractMin(Q)
        for all vertex v 2 u:AdjacencyList do
            if d[v] > d[u] + w(u; v) then
                d[v]   d[u] + w(u; v)
```

At each iteration, the algorithm expands the node with the shortest distance and updates distances to all reachable nodes. The algorithm terminates when there are no updates in the distance variables of the graph.

Answer for small-graph data :

```
1       0 2:3:

2       1 3:4:

3       1 2:4:5:

4       10000 5:

5       10000 1:4:
```

Answer for large graph data:

```
1       0 2:
2       1 3:
3       2 2:13:
4       5 14:
5       5 14:
6       10 7:
7       9 6:8:17:
8       10 7:20:
9       13 10:
10      12 9:20:
11      5 12:
12      4 11:13:22:
13      3 3:12:14:
14      4 4:5:13:15:
15      5 14:16:
16      6 15:27:
17      8 7:27:
18      13 19:35:
19      12 18:20:21:
20      11 8:10:19:
21      13 19:
22      5 12:24:
23      8 32:
24      6 22:32:
```

25	8 26:32:
26	8 25:27:
27	7 16:17:26:28:33:
28	8 27:44:
29	17 37:38:
30	16 45:
31	8 32:
32	7 23:24:25:31:40:41:46:
33	8 27:42:43:
34	17 49:
35	14 18:50:
36	16 50:
37	18 29:
38	16 29:45:53:
39	16 45:
40	8 32:55:
41	8 32:47:
44	9 28:43:52:
45	15 30:38:39:61:
46	8 32:56:60:
47	9 41:
48	10 43:
49	16 34:50:71:
50	15 35:36:49:58:
51	17 58:
52	10 44:
53	17 38:62:
54	11 66:

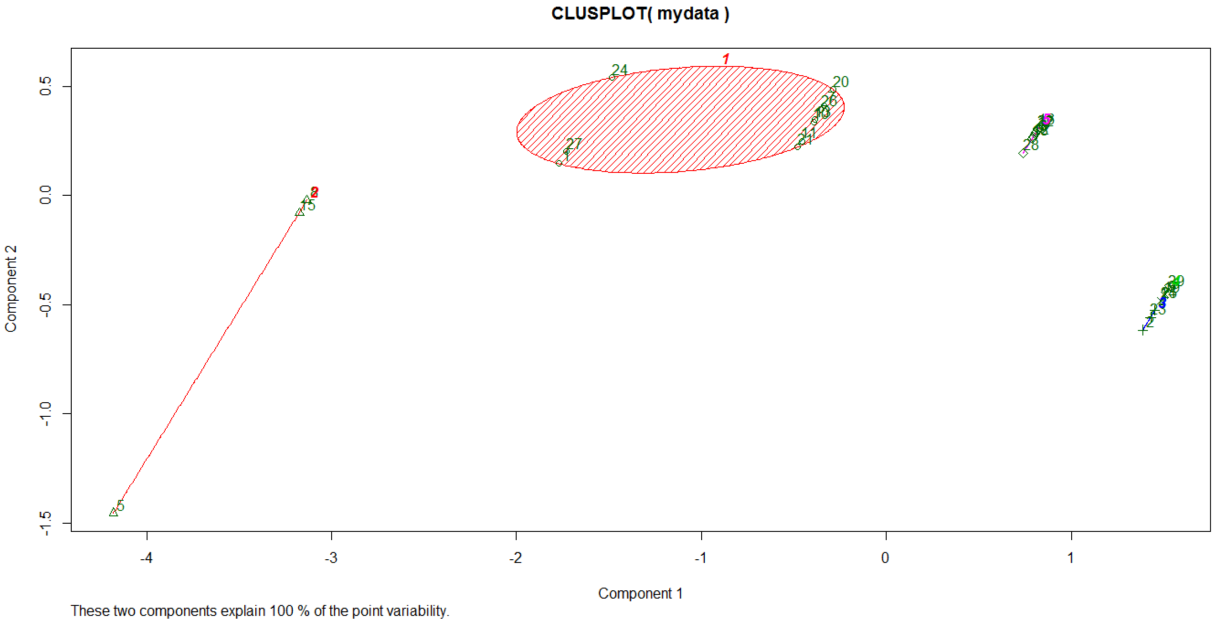
55	9 40:
56	9 46:68:
57	13 43:69:
58	16 50:51:72:
59	13 69:
60	9 46:66:67:
61	14 45:65:
62	18 53:63:64:
63	19 62:
64	19 62:
65	13 61:80:
66	10 54:60:79:
67	10 60:
68	10 56:77:78:
69	12 57:59:77:
70	13 82:102:
71	17 49:72:
72	17 58:71:73:74:75:
73	18 72:
74	18 72:
75	17 72:76:104:
76	18 75:
77	11 68:69:82:98:
78	11 68:81:
79	11 66:80:81:
80	12 65:79:89:
83	18 85:
84	18 85:

```
85      17 83:84:86:

86      16 85:87:

87      15 86:88:

88      14 87:89:91:

89      13 80:88:90:

90      14 89:93:

91      15 88:92:

92      16 91:93:

93      15 90:92:94:

94      14 93:95:96:

95      13 81:94:

96      15 94:

97      14 99:

98      12 77:99:

99      13 97:98:100:

100     14 99:107:

101     13 82:105:

102     14 70:103:

103     15 102:104:105:

104     16 75:103:106:

105     14 101:103:107:

106     17 104:108:

107     15 100:105:

108     18 106:109:110:

109     19 108:

110     19 108:
```

The detailed java Hadoop implementation and code can be found in the file, “ShortestPath.zip”.

K Means Clustering

With 5 clusters


CLUSPLOT(mydata)
Component 2
Component 1
These two components explain 100 % of the point variability.