# Big Data Content Retrieval, Storage And Analysis Foundations Of Data Intensive Computing

**Chetan Pachchhao, Sushant Deshpande**
**State University of New York, Buffalo**
**Contact: chetanpa@buffalo.edu , sushants@buffalo.edu**

**Project Objectives**

Project Title project will meet the following objectives.

- Learning and using map reduce concepts for data analysis using hadoop.
- Recognize a data-intensive problem.
- Analyze the data requirements of a problem.
- Clean the raw data and us it for processing.
- Implement hadoop framework using java apply map reduce strategy for analysis of large scale data.

**Project Approach**

We are implementing project for aggregating twitter data and analyzing it to find trends and patterns of popular social micro blogging. People use social media communication a lot. Twitter allows us to hash tagging to stress particular details. These can be used to collect data specific to a topic or domain. Also it provides us info about how many people are interested in particular domain/person by allowing us to find the number of followers. We first collect and clean the data using public streams of twitter. It is done using twitter api's. These trends and patterns can be analyzed to use for marketing or business purposes.

**Hadoop:**

It is the open source platform for storage and management of large amount of data. It stores data sets across distributed set of clusters of servers and uses distributed analysis approach in each cluster. It is highly scalable and can be scaled to thousands of machines.

**Map-Reduce:**

It is a programming paradigm that performs operations on data using two different jobs.

Map component (function) takes the data breaking those into <key, value> pair format. The reduce component (function) takes the output of map function as input and combines the values into smaller sets according to keys.

We have implemented following things in the project:

- **Finding trends**
  This includes finding word count, most trending word count, most trending hash tag count, most trending @ count. The mapper tokenizes the words from the tweets and sends these words to reducer. Reducer counts the frequency of each word in the whole document.
- **Finding Co-occurring hashtags**
  If a tweet contains more than one hash tag then co-occurring hash tag is combinations of each hash tag with all other hash tags in that tweet. Mapper sends such key value

pair to reducer and reducer counts the number of co-occurring hash tags. There are two different approaches:

Pair approach
Stripes approach

In pair approach we send each hash tag pair to reducer. In stripes approach we considered only one hash tag and put other hash tags into a map and sent hash tag, map pair to reducer.

In pair approach we used partitioner and two reducers. Output produced by mapper is sent to partitioner and partitioner decides to which reducer this mapper output should be sent.

- **Kmeans clustering:**

  We have counted the number of followers of each user. We have divided the data into 3 groups with lower number of followers, medium number of followers, high number of followers.

  This gives us idea about how popular the person is on twitter. Also we have found the median value of number of followers for each group.

- **Dijkstra's Algorithm**
  Find the shortest path for each node from the given source node using map-reduce approach.

**Algorithms:**

**Simple Word Count :**

```
class Mapper
        Tokenize the tweet into list of words.
        method Map(lineid a, tweet tw)
                for all words w ∈ tweet tw do
                        Emit(word w, count 1)

class Reducer
        method Reduce(word w, counts [c1 , c2 , . . .])
                sum ← 0
                for all count c ∈ counts [c1 , c2 , . . .] do
                        sum ← sum + c
                        Emit(word, count sum)
```

Output sample:

| | |
|---|---|
| PepsiIPL2014 | 56 |
| PepsiIPL7%2014 | 9 |
| Pepsi_IPL2014 | 17 |
| Pepsi_IPL7_Live | 1 |
| Pepsi_IPL_2014 | 14 |
| Pepsi_IPL_7 | 2 |
| Pepsus%Alalius | 4 |
| Per | 2 |
| Percy | 1 |

**Most trending words :**

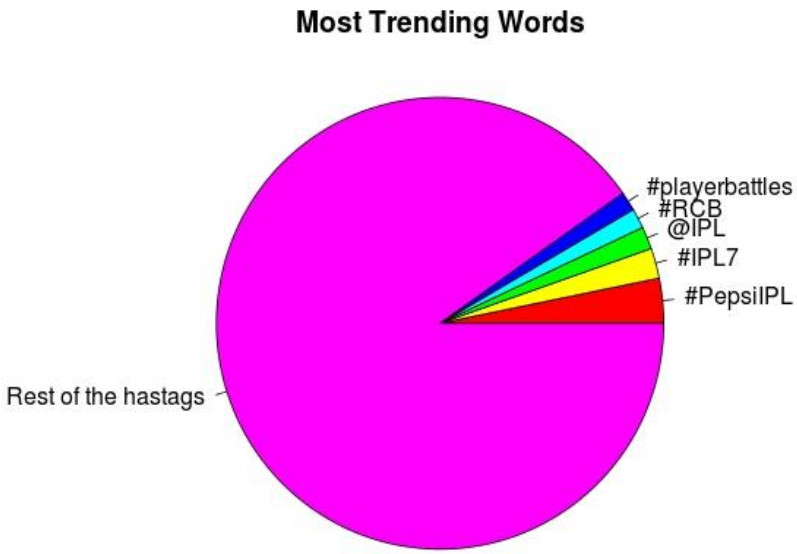
```
class Mapper
        Tokenize the tweet into list of words.
        method Map(lineid a, tweet tw)
                for all words ∈ tweet tw do
                        Emit(word w, count 1)
```

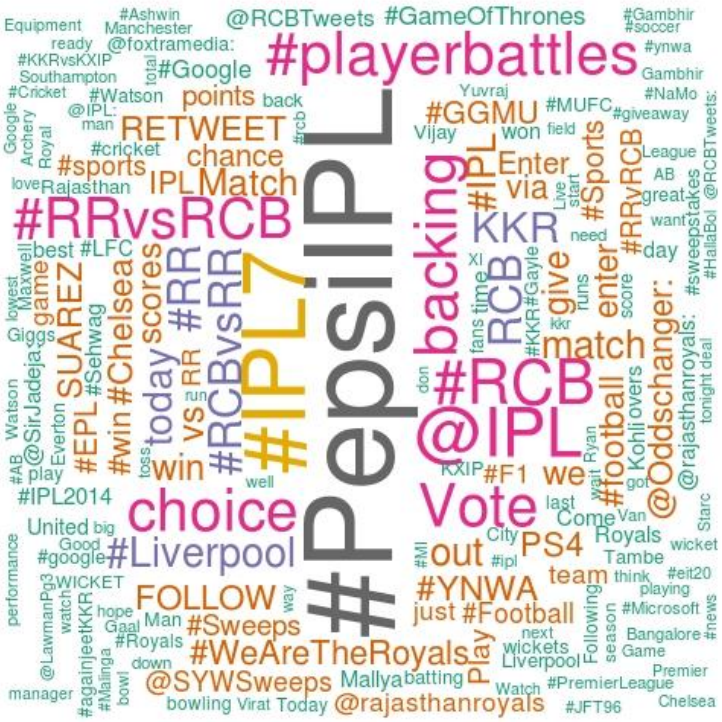
```
class Reducer
        method Reduce(word w, counts [c1 , c2 , . . .])
                sum ← 0
                for all count c ∈ counts [c1 , c2 , . . .] do
                        sum ← sum + c
                        Put w and sum into different arraylists.

        method Cleanup(Context contex)
                while arraylist of sum is not empty do
                        i = index of maximum element in arraylist of sum
                        emit(arraylist of words[i],arralist of count[i])
                        remove arraylist of words[i]
                        remove arralist of sum[i]
```

Sample Output:

| | |
|---|---|
| #PepsiIPL | 10045 |
| #IPL7 | 6758 |
| @IPL | 5022 |
| #RCB | 4448 |
| #playerbattles | 4305 |
| choice | 4278 |
| Vote | 4234 |






Most Trending Words

**Hashtag Count**

class Mapper

Tokenize the tweet into list of words.

method Map(lineid a, tweet tw)

for all words w ∈ tweet tw do

if tw contains #

Emit(word w, count 1)

class Reducer

method Reduce(word w, counts [c1 , c2 , . . .])

sum ← 0

for all count c ∈ counts [c1 , c2 , . . .] do

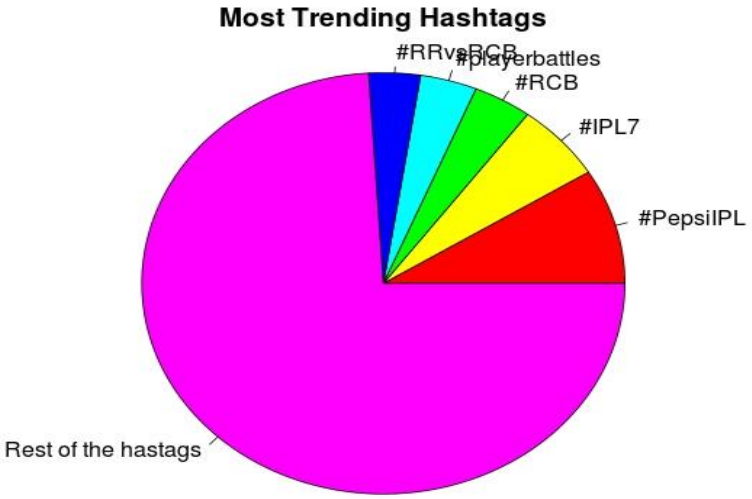
sum ← sum + c

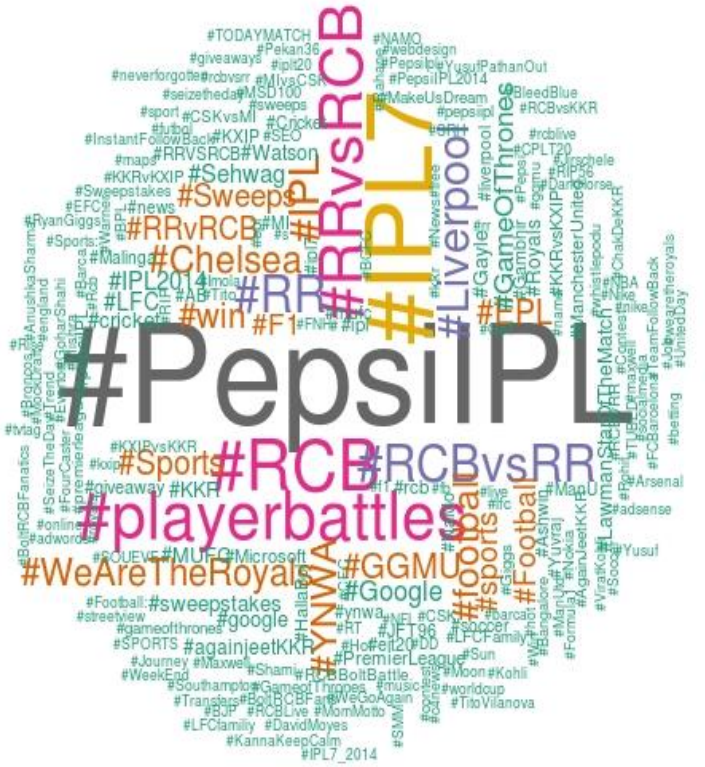
Emit(word, count sum)

Output sample:

| | |
|---|---|
| #PepsiIPL | 10045 |
| #IPL7 | 6758 |
| #RCB | 4448 |
| #playerbattles | 4305 |

| | |
|---|---|
| #RRvsRCB | 3911 |
| #RCBvsRR | 3000 |
| #RR | 2926 |



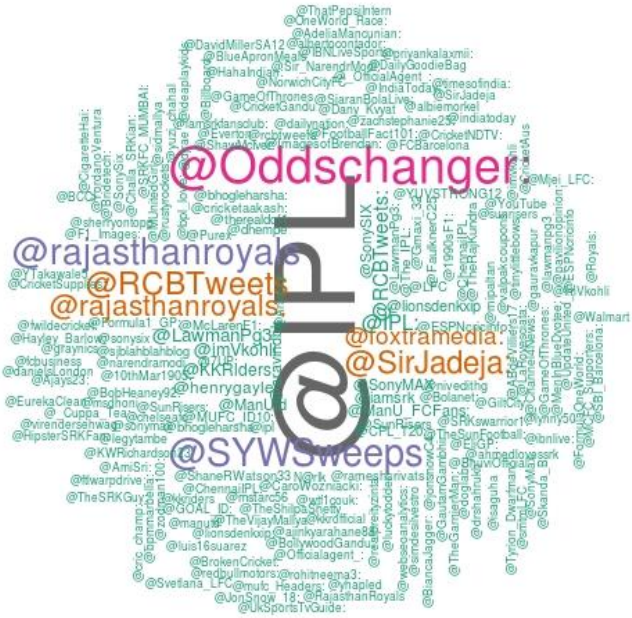

Most Trending Hashtags

**@ Count :**

```
class Mapper
        Tokenize the tweet into list of words.
        method Map(lineid a, tweet tw)
                for all words w ∈ tweet tw do
                        if tw contains #
                                Emit(word w, count 1)
class Reducer
        method Reduce(word w, counts [c1 , c2 , . . .])
                sum ← 0
                for all count c ∈ counts [c1 , c2 , . . .] do
                        sum ← sum + c
                        Emit(word, count sum)
```

Sample Output:

| | |
|---|---|
| @IPL | 5022 |
| @Oddschanger: | 1922 |
| @SYWSweeps 1324 | |
| @rajasthanroyals | 1268 |
| @RCBTweets | 1040 |
| @SirJadeja: | 995 |
| @rajasthanroyals: | 888 |




Most Trending @

**Co-occurring Hashtags Pair approach:**

```
class Mapper
        method Map(lineid a, tweet tw)
                Tokenize the tweet into list of words.
                for all words do
                        if word w contains #
                                put w into arralist of hashtags
                Sort arralist of hashtags.
                for i=0 to length of arraylist of hashtags do
                        for j=i+1 to length of arraylist of hashtags do
                                emit(pair(arraylist[i],arralist[j]), count 1)

class Reducer
        method Reduce(pair p, counts [c1 , c2 , . . .])
                sum ← 0
                for all count c ∈ counts [c1 , c2 , . . .] do
                        sum ← sum + c
                        Emit(pair p, count sum)
class Partitioner
        method getPartition(key pair, value count)
                if pair starts with letter A-M
                        return 0
                else
                        return 1
```

Sample Output:

```
Partitioner 0
#BALL #Eit20   1
#BALL #PepsiIPL        1
#BALL #RCB             1
#BALL #RCBvRR          1
#BANG #PepsiIPL        1
#BANGLE #BOHO          1

Partitoner 1
#Virat #YuvrajSingh    1
#Virat #eit20          1
#Virat #pepsiIPL       2
#ViratKohli #eit20     2
#ViratKohli #onemanarmy     2
```

**Co-occurring hashtags stripes approach :**

```
class Mapper
        method Map(lineid a, tweet tw)
                Tokenize the tweet into list of words.
                for all words do
                        if word w contains #
                                put w into arralist of hashtags
                Sort arralist of hashtags.
                for i=0 to length of arraylist of hashtags do
                                for j=i+1 to length of arraylist of hashtags do
                                        if Map contains hashtag[j]
                                                hashtag[j].count =  hashtag[j].count +1
                                        else
                                                put hashtag[j] in to Map
                                                set hashtag[j].count =1
                                emit(hashtag[i],Map)


class Reducer
        method Reducer(word w, Map[m1 , m2 . . .])
                for all Map m ∈ Map[m1 , m2 . . .] do
                        for all entries in m
                                if OutputMap contains m.key
                                        OutputMap.key.count = OutputMap.key.count +
m.key.count
                                        update count of key in OutputMap
                                else
                                        Put m.key and m.key.count into OutputMap

                for all entries output in OutputMap do
                        emit(pair (word ,output.key ), output.key.count)
```

Sample Output:

| | |
|---|---|
| #AmazonHappyHour_#pepsiipl | 1 |
| #AmazonHappyHour_#ipl | 1 |
| #Ambani_#mi | 1 |
| #America_#Iraq | 4 |
| #America_#Sports | 1 |
| #America_#androidgames | 1 |

**Co-occurring hashtag relative frequency Pair approach** :

```
class Mapper
        method Map(lineid a, tweet tw)
                Tokenize the tweet into list of words.
                for all words do
                        if word w contains #
                                put w into arralist of hashtags
                Sort arralist of hashtags.
                for i=0 to length of arraylist of hashtags do
                        count ←0
                        for j=i+1 to length of arraylist of hashtags do
                                emit(pair(arraylist[i] ,arralist[j]), count 1)
                                count ← count +1
                        emit (pair(arraylist[i] ,#* ), count)
class Reducer
        totalcount ← 0
        method Reduce(pair p, counts [c1 , c2 , . . .])
                sum ← 0
                for all count c ∈ counts [c1 , c2 , . . .] do
                        sum ← sum + c
                if pair p contains *
                        totalcount ← sum
                else
                        emit(pair p , (sum/totalcount))
class Partitioner
        method getPartition(key pair, value count)
                if pair starts with letter A-M
                        return 0
                else
                        return 1
```

Sample Output:

```
Partitioner 0
#BALL #Eit20    0.25
#BALL #PepsiIPL         0.25
#BALL #RCB      0.25
#BALL #RCBvRR           0.25
#BANG #PepsiIPL         1.0

Partitioner 1
#QPRvMIL #stayingup 0.14285715
#QPRvMIL #stream            0.14285715
```

#QPRvMIL #togetherRs 0.14285715
#QSL #Qatar_Cup 0.5
#QSL #football 0.5

**Co-occurring hashtag relative frequency Stripes approach** :

```
class Mapper
        method Map(lineid a, tweet tw)
                Tokenize the tweet into list of words.
                for all words do
                        if word w contains #
                                put w into arralist of hashtags
                Sort arralist of hashtags.
                for i=0 to length of arraylist of hashtags do
                                for j=i+1 to length of arraylist of hashtags do
                                        if Map contains hashtag[j]
                                                hashtag[j].count =  hashtag[j].count +1
                                        else
                                                put hashtag[j] in to Map
                                                set hashtag[j].count =1
                                emit(hashtag[i],Map)

class Reducer
        method Reducer(word w, Map[m1 , m2 . . .])
                totalcount ← 0
                for all Map m ∈ Map[m1 , m2 . . .] do
                        for all entries in m
                                if OutputMap contains m.key
                                        OutputMap.key.count = OutputMap.key.count +
m.key.count
                                        update count of key in OutputMap
                                         totalcount ← totalcount + m.key.count
                                else
                                        Put m.key and m.key.count into OutputMap
                                        totalcount ← totalcount + m.key.count

                for all entries output in OutputMap do
                        emit(pair (word ,output.key ), (output.key.count)/totalcount)
```

Sample Output:

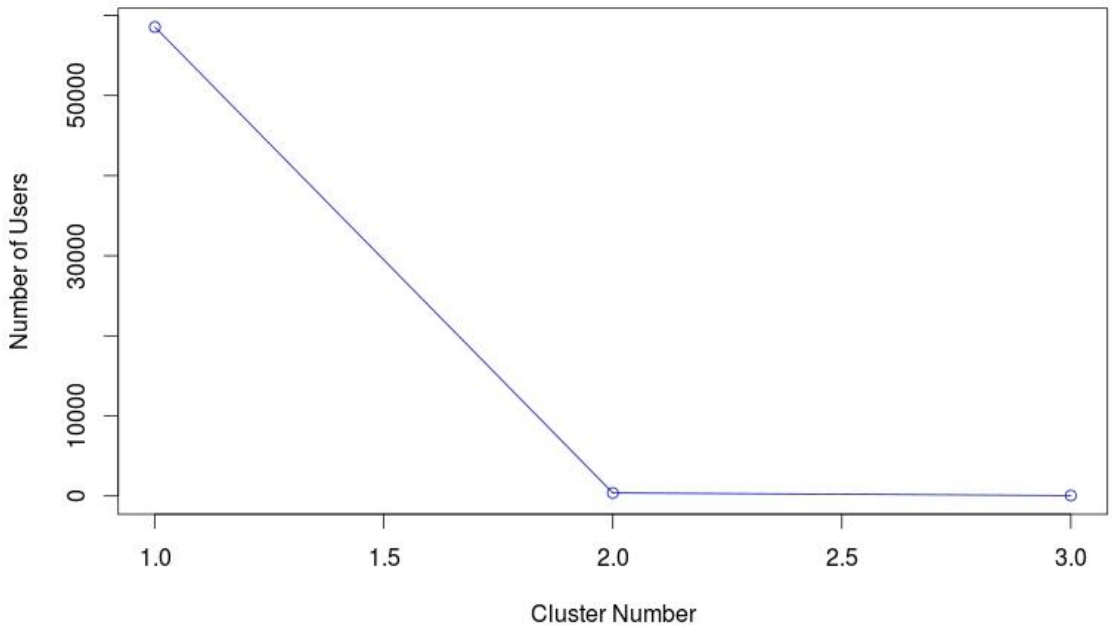
#AWESOME_RED_#TOPKOPITE 0.5
#AWESOME_RED_#YNWA 0.5

#AXNAsia_#Watson: 0.33333334
#AXNAsia_#Sherlock: 0.33333334
#AXNAsia_#NowScreening 0.33333334

**Kmeans**

```
class mapper:
        method map(lineid a, tweet tw)
                Tokenize the tweet and find follower count
                Find the difference of follower count with each median and add it to the bucket
of                      median with minimum difference.
class Reducer
        method Reducer(key groupid, value[f1,f2,...])
                sum = 0
                count = 0
                for each value f in value[f1,f2,...]
                        sum = sum + f
                        Increment count
                NewMedian  = sum/count
                emit(groupid, NewMedian)  // write new median into output file of reduce

class Kmeans
        method run()
                Initialize the medians.
                While(! isDone)
                        get median from output file of reducer into Imap
                        if _Map is empty     // 1st iteration
                                Assign values of Imap to _Map
                        else
                                if _Map and Imap contain same values
                                        set isDone
                                else
                                        Assign values of Imap to _Map
```




Medians:
1 429
2 342687
3 1389004

**Dijkstra :**

class mapper:
    method map(lineid a, line l)
        Tokenize the line l
        Get current node id, distance from source and its neighbors.
        Increment distance = distance + 1.
        For each neighbor n
            Get n.id
            Assign n.distance = distance
            emit(n.id,pair("distance",n.distance))
        emit(id, distance-1)
        emit(id, pair("nodes",List of neighbours))

        Nodes 2:3:4
        value 24

```
Class reducer
        method  reducer (node n,Text distanceInfo )
                set lowest to infinity
                for each distanceInfo d
                        if d contains nodes // distanceInfotains list of neighbors
                                nodeList ← list of neighbors
                        else // distanceInfo contains distance
                                dist = distance // received in distancInfo
                                lowest = min(dist,lowest)
                emit(nodeid , pair(lowest,nodeList))


Class Dijkstra
        method run()
                While(! isDone)
                        get node details from output file of reducer into Imap
                        if _Map is empty     // 1st iteration
                                Assign values of Imap to _Map
                        else
                                if _Map and Imap contain same values
                                        set isDone
                                else
                                        Assign values of Imap to _Map
```

Sample Output on large 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:
42 9 33:
43 9 33:44:48:
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:
81 12 78:79:95:
82 12 70:77:101:
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:

Sample Output on small data:

1 0 2:3:
2 1 3:4:
3 1 2:4:5:
4 2 5:
5 2 1:4:

**Lessons Learned**

Large data sets can be efficiently analyze and patterns and trends related to these can be found out using map-reduce algorithms in hadoop.