

DATA INTENSIVE COMPUTING

CSE 587

PROJECT 2

DATA ANALYTICS USING HADOOP/MAP REDUCE FRAMEWORK

ANKIT JAIN #50097432

MILKY SAHU #50096350

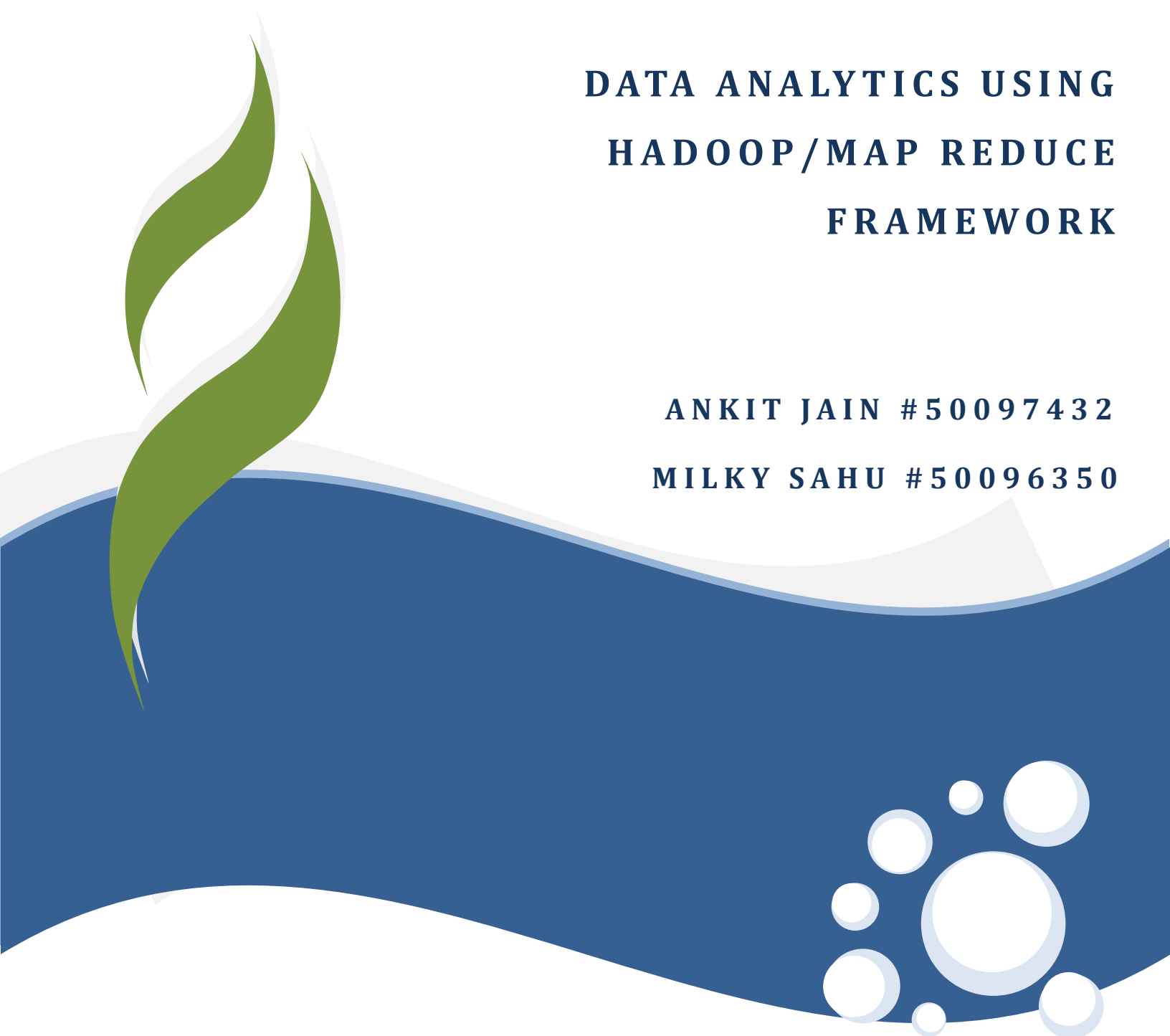




TABLE OF CONTENTS

» Abstract	
» Project Objectives	
» Project Approach	
» Data Information	
» Data format and source	
» Amount of data collected and details	
» Aggregator details, APIs used	
» MR (mapper, reducer pseudo code) programs	
» Configuration of the cluster used	
» Outputs: different outputs and visualizations	
» References	



Data Analytics using Hadoop/MapReduce framework

Abstract

In order to aggregate interesting data and also to keep up with the “trends” we will aggregate data from Twitter for different ranges of dates (week-range, month-range). Aggregated raw data needs to be cleansed to some extent before analyzing it using Big-data methods. In this project-

- We want to find out the most trending words.
- We are also interested in finding out co-occurring hash tags.
- We want to cluster the tweeters by the number of followers they have. We will need three clusters where the average number of followers is low, medium and high respectively, where the actual average value for each cluster will depend on the data size. This information may be used for marketing and for targeting the ads/messages.
- For the most popular hash tag, we want to develop a connected network of sender → receiver pattern, label the network for determining the shortest path between nodes of this network.

Project Objectives

Data Analytics using Hadoop/MapReduce framework will meet the following objectives:

- Help understand the components and core technologies related to content retrieval, storage and data-intensive computing analysis.
- Enable to explore designing and implementing data-intensive (Big-data) computing solutions, using MapReduce (MR) programming model.
- Using Hadoop 2 for HDFS and MR infrastructure
- Teach visualize the data using appropriate tools.

Project Approach

- Reviewed the Java/Python language skills by working on the given sample application.
- Understood the MR model and modeling the data as <key, value> store.
- Installed Hadoop 2 on the machine and tested it out.
- Ran the wordcount program on the laptop and understood the code for the Mapper, Reducer, and the main MR job configuration.
- Data Aggregator: Explored ways of aggregating twitter data, chose the topic, tried some of the existing APIs for twitter available at <https://dev.twitter.com/docs>; collected data. We used Twitter4J.
- Tweet Collection: Cleaned the unwanted details from the tweets and saved them in a set of regular files in a designated directory.
- Designed and implemented the various MR workflows to extract various information from the data. (i)simple wordcount (ii)trends (iii)#tag counts (iv)@xyz counts etc.
- Continued on to the word-co-occurrence algorithm, both “pairs” and “stripes” approach.
- Used each tweet as the context for determining the neighbors in the algorithm.
- Next we moved on to clustering: used Mapreduced version of K-means.
- Performed visualization of data.



Data Information

Our objective is to find the trends in latest mobile cellular phones from various leading companies. We aim to **figure out the biggest competitors of** upcoming and much awaited **iPhone6** from apple.

The rival mobile phones from different companies are:

Apple – iPhone5s

Samsung – Galaxy S5, Galaxy Note 3, Galaxy Note 2

Nokia – Lumia 1320, Lumia 930, Lumia 635, Lumia 630

Google – Nexus 5

HTC – HTC One

For this we have collected several tweets on the basis of **their latest hashtags and handles** such as:

#HTCOneM8	#lumia930	@GoogleNexus	Google Nexus 5 Review
#HTC1	#lumia635	Samsung Galaxy S5	#GoogleNexus5Camera
HTCOne	#lumia1020	LG Google Nexus 5	#GoogleNexus5Features
HTC	Galaxy S5	#iphone6	Samsung Galaxy S5 Camera
#sfive	GalaxyS5	#S5, #s5	Nexus Google Phone
#nexus4	iPhone6	#samsunggalaxys5	HTCOne vs GoogleNexus
iPhone News	#Iphone5S	#Lumialcon	Samsung Galaxy 5
#nexus5	NokiaLumia	#GALAXYNote3	Samsung Galaxy S5 vs. Apple iPhone 5S
#iphonesix	#iphonefive	#galaxynotetwo	#galaxynotethree
#galaxyNote3	#GalaxyS5Info	#TEAMiPHONE	@Smartphones

- **Data Source:** <http://www.twitter.com/> (Twitter4J)
- **Amount of Data collected:** We collected over 65,00,00 tweets.

Data Format

The data along with the tweets gives other pieces of information like **retweet count**, **time created at**, **favorite count**, **geographic location**, **place** etc. as shown in the code snippet below:

```
@Override
public void onStatus(Status status) {
    String totalTweetText = status.getText() + " , " + status.getRetweetCount() + " , "
        + status.getCreatedAt() + " , " + status.getFavoriteCount() + " , "
        + status.getGeoLocation() + " , " + status.getPlace();
    if(isEnglish(totalTweetText)) {
        LOG.info(totalTweetText);
    }
}
```

The data when fetched looks like below. The highlighted parts show the different attributes of the tweets obtained:

```
Meet Me Downtown #lumia1020 #nofilter #brooklyn #boerumhill @ Brooklyn Fare http://t.co/YUvv8QH0IK , 0 , Sun Apr 20 20:34:44 EDT 2014 , 0 ,
GeoLocation(latitude=40.68898893, longitude=-73.98586962) , PlaceJSONImpl{name='New York', streetAddress='null', countryCode='US', id='27485069891a7938',
country='United States', placeType='city', url='https://api.twitter.com/1.1/geo/id/27485069891a7938.json', fullName='New York, NY', boundingBoxType='Polygon',
boundingBoxCoordinates=[[Ljava.lang.Object;@38f52df8], geometryType='null', geometryCoordinates=null, containedWithinIn=[]}
iPhone 6 Rumors and News: Larger front panel leaked, and a possible price increase looms http://t.co/OwMMOeGWPe , 0 , Sun Apr 20 20:34:57 EDT 2014 , 0 ,
GeoLocation(latitude=33.67806986, longitude=-86.61374331) , PlaceJSONImpl{name='Alabama', streetAddress='null', countryCode='US', id='288de3df481163e8',
country='United States', placeType='admin', url='https://api.twitter.com/1.1/geo/id/288de3df481163e8.json', fullName='Alabama, USA', boundingBoxType='Polygon',
boundingBoxCoordinates=[[Ljava.lang.Object;@689d11fe], geometryType='null', geometryCoordinates=null, containedWithinIn=[]}
```

Aggregator Details, APIs used

We have used **Twitter4J** aggregator. Twitter4J is an unofficial Java library for the Twitter API. With Twitter4J, we could easily integrate our Java application with the Twitter service. Just by adding twitter4j-core-4.0.1.jar to our application classpath we could make use of this library.

System Requirements:

OS: Windows or any flavor of Unix that supports Java.
JVM: Java 5 or later

Twitter4J includes software from JSON.org to parse JSON response from the Twitter API.

The classes such as Driver.java and EnglishListenerClass.java were modified to fetch the desired tweets. Driver class on running collected tweets in a text file. This text file was appended with more tweets every time the program was run. Along with the tweets from desired hashtags and handles, some informations about the tweets could also be fetched using the getter methods in the library such as `getText()`, `getRetweetCount()`, `getCreatedAt()`, `getFavoriteCount()`, `getPlace()`, `getGeoLocation()` etc.

MR (mapper, reducer pseudo code) programs

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*. The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

We have worked on four methods of data analysis based on map reduce:

- **Word Count**
- **Co-occurrence**
- **K-means clustering**
- **Shortest path**

Word Count:

WordCount is a simple application that counts the number of occurrences of each word in a given input set. The word count operation takes place in two stages a mapper phase and a reducer phase. In mapper phase first the text is tokenized into words then we form a key value pair with these words where the key being the word itself and value '1'. For example consider the sentence "tring tring the phone rings". In map phase the sentence would be split as words and form the initial key value pair as

<tring,1>

<tring,1>

<the,1>

<phone,1>

<rings,1>

In the reduce phase the keys are grouped together and the values for similar keys are added. So here there are only one pair of similar keys 'tring' the values for these keys would be added so the output key value pairs would be

<tring,2>

<the,1>

<phone,1>

<rings,1>

This would give the number of occurrence of each word in the input. Thus reduce forms an aggregation phase for keys.

Algorithm:

The mapper emits an intermediate key-value pair for each word in a document.

The reducer sums up all counts for each word.

1: class Mapper

2: method Map(docid a; doc d)

3: for all term t 2 doc d do

4: Emit(term t; count 1)

1: class Reducer

2: method Reduce(term t; counts [c1; c2; :::])

3: sum 0

4: for all count c 2 counts [c1; c2; :::] do

5: sum sum + c

6: Emit(term t; count sum)


```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
#Spark 1  
#SpeakUN2014 2  
#SpecialOffer 6  
#SpecialReports 14  
#SpecialUN 2  
#Specification 59  
#SpeckCandyShellGripCase 60  
#Spencer 2  
#Spenzo 2  
#Spica 4  
#SpiderMan 282  
#SpiderMan. 2  
#SpiderMan2 1  
#SpiderWoman 2  
#Spiderman 9  
#Spiderman!! 470  
#SpidermanCanal5 6  
#SpiffykermsOfficeStyle 2  
#Spigen 3  
#SpikeEnBellosAnimales 6  
#SpinnrTaylorSwift 22  
#Spitter 2  
#Spoller 4  
#Spollt 1  
#SpongeBob 1  
#Sponsored 24  
#Sport 6  
#SportMode 2  
#SportSatu 6  
#Sports 4  
#SportsRoadhouse 2  
#Sportscenter 2  
#Spotify 13  
#SpreadtheLove 2  
#Spring 14  
#SpringBreakGoals 2  
#SpringClassic 2  
#SpringDecor  
^Chduser@ajain22-Inspiron-5521:~$
```

Word Count

```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
#retweetthis 2  
#retweettt 2  
#reuters 16  
#revealed 3  
#revenge 44  
#revenge! 2  
#revenge? 2  
#revenue 1  
#reverbNation 2  
#reverse 2  
#review 148  
#review! 4  
#review. 2  
#review... 5  
#review: 6  
#reviews 39  
#revine 2  
#revise 1  
#revision 5  
#revisionpowerhour 1  
#revisions 2  
#revolt 2  
#revolution 8  
#revues 2  
#rey 2  
#rffidyourbusiness 3  
#rfp: 2  
#rghoz 2  
#rgndi 3  
#rharebreed 2  
#rheysapowerbotton 2  
#rheingold 2  
#rhianna. 2  
#rhinestones! 1  
#rhoa 142  
#rhoa" 2  
#rhoa. 2  
^C  
hduser@ajain22-Inspiron-5521:~$
```

Hashtags

```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
@flipadelphia: 2  
@flipagram 2  
@flipbeans 1  
@flipboard 8  
@flipdiapers 2  
@flipkart 36  
@flipkart. 4  
@flipkart..deleting 4  
@flipkartsupport 13  
@flipn_bricks 2  
@flipp_2: 2  
@flipper3000 2  
@flippermagz: 2  
@flirtingtext: 2  
@flyflyanlaa: 2  
@flarrain: 2  
@flmss 4  
@flo0r28 2  
@flo_madd: 2  
@flogalo 2  
@flomohorton 2  
@flooksrtaan 4  
@floodooooorb 2  
@floodorbieber_: 2  
@floodoriane_jr: 3  
@floodoroviedo: 2  
@floodplinnaa: 2  
@floor_swaggy 2  
@floorcitabaier: 2  
@floorgarcia01: 2  
@flooribarra16 2  
@flooribarra16: 2  
@floorlotacono 2  
@floorlunna97 2  
@floorvaraa 2  
@floorvital98: 2  
@floorzacarias07: 2  
@flopaderont^C  
hduser@ajain22-Inspiron-5521:~$
```

Handles

```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
hduser@ajain22-Inspiron-5521:~$ cat part-r-00000 | head  
#iphone 19934  
#galaxys5. 17736  
#galaxys5 16709  
#gameinsight 9757  
#votekatniss 8968  
#android 8939  
#htconem8 8881  
#iphonegames, 8249  
#votetris 8220  
#iphone6 7406  
hduser@ajain22-Inspiron-5521:~$
```

Trends

Co-occurrence:

This program generates a matrix containing the relative frequencies of a pair of words co-occurring with each other in a paragraph. A co-occurrence matrix could be described as the tracking of an event, and given a certain window of time or space, what other events seem to occur. For the purposes of this post, our “events” are the individual words found in the text and we will track what other words occur within our “window”, a position relative to the target word.

Pairs

Implementing the pairs approach is straightforward. For each line passed in when the map function is called, we will split on spaces creating a String Array. The next step would be to construct two loops. The outer loop will iterate over each word in the array and the inner loop will iterate over the “neighbors” of the current word. The number of iterations for the inner loop is dictated by the size of our “window” to capture neighbors of the current word. At the bottom of each iteration in the inner loop, we will emit a WordPair object (consisting of the current word on the left and the neighbor word on the right) as the key, and a count of one as the value.

Algorithm:

Class MAPPER

Method MAP(docid a, doc d)

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

for all term $u \in \text{NEIGHBORS}(w)$ do

EMIT(pair (w,u), count 1)

Class REDUCER

Method REDUCE(pair p, counts[c1, c2,...])

$s \leftarrow 0$

for all count $c \in \text{counts}[c1, c2,...]$ do

$s \leftarrow s+c$

EMIT(pair p, count s)

Stripes

Implementing the stripes approach to co-occurrence is equally straightforward. The approach is the same, but all of the “neighbor” words are collected in a HashMap with the neighbor word as the key and an integer count as the value. When all of the values have been collected for a given word (the bottom of the outer loop), the word and the hashmap are emitted.

Algorithm:

Class MAPPER

Method MAP(docid a, doc d)

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

$h \leftarrow \text{new ASSOCIATIVEARRAY}$

for all term $u \in \text{NEIGHBORS}(w)$ do

$H\{u\} \leftarrow H\{u\}+1$

EMIT(TERM w, STRIPE H)

Class REDUCER

Method REDUCE(term w, stripes [H1, H2, H3,...])

$H_f \leftarrow \text{new ASSOCIATIVEARRAY}$

for all stripe $H \in \text{stripes [H1, H2, H3,...]}$ do

SUM(H_f , H)

EMIT(term w, stripe H_f)

```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
-fullNameNeill 5  
-fullNameOak 5  
-fullNameOakland 7  
-fullNameOatfield 7  
-fullNameOberkochen 4  
-fullNameOgden 1  
-fullNameOhio 73  
-fullNameOkanaganSimilkameen 4  
-fullNameOklahoma 64  
-fullNameOldham 8  
-fullNameOlympia 2  
-fullNameOmaha 15  
-fullNameOman 23  
-fullNameOntario 6  
-fullNameOpalocka 10  
-fullNameOpelika 4  
-fullNameOrange 18  
-fullNameOregon 8  
-fullNameOrem 5  
-fullNameOrlando 15  
-fullNameOrtiz 2  
-fullNameOslo 9  
-fullNameOsseo 1  
-fullNameOttawa 12  
-fullNameOtttery 9  
-fullNameOuter 4  
-fullNameOxford 11  
-fullNameOxnard 5  
-fullNamePaddington 41  
-fullNamePademangan 6  
-fullNamePadova 4  
-fullNamePakistan 66  
-fullNamePalatka 4  
-fullNamePalm 10  
-fullNamePalmdale 21  
-fullNamePalmetto 4  
-fullNamePalo 10  
-fullNamePalos^C  
hduser@ajain22-Inspiron-5521:~$
```

Pairs

```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
Christylane null:1.0  
Chrith null:1.0  
ChrnnisaS :1.0  
ChromeWireless null:1.0  
Chromebox null:1.0  
Chronometer GameInsight:1.0  
Chrris null:1.0  
Chrysler :1.0  
ChubbsDeckert indeed:1.0  
ChuchuAdrian :1.0  
Chuck ClariSepturca:0.1 null:0.1 httpcoSejaLhFu:0.2 httpcoeguMBcjAg:0.2 httpcoASXkoWu:0.1 gossipgirl:0.1 Blair:0.1 perfect:0.1  
ChuckCheesman BorderSongsCD:1.0  
ChuckNorrisRR vida:1.0  
ChuckTaylorDude null:1.0  
Chuckaduk null:1.0  
Chuffs it:1.0  
Chuggington httpcoSoMBmFXfi:1.0  
ChummyPandaNews D:1.0  
Chums null:1.0  
ChunAlan null:1.0  
Chunti httpstcoibbAqNZL:1.0  
Chupa hahaha:0.3333333333333333 Lannister:0.3333333333333333 Jjjajaja:0.3333333333333333  
ChupaSantos santos:1.0  
ChurchOfBarca though:1.0  
Chutch Meg:0.6666666666666666 PLS:0.3333333333333333  
Chuuuuuuuuuuuuhaaaaaaaaporque JAAJJAJA:1.0  
Chuva ahuaha:1.0  
Chuys null:1.0  
Chw kune:1.0  
Ciao uds:0.5 tutti:0.5  
Cibe httpcoMBKjxHmE:1.0  
CibeZ null:1.0  
CiciliaWP pls:1.0  
Cicis :1.0  
Cicpc httpcoHxOzrduXY:1.0  
Cid gark:1.0  
CidBabel httpcoUzEfLrKS:1.0  
Ctee hahha:0.2 :0.2 hahhahh:0.2 Cleeeee:0.2 perjuangan:0.2  
^Chduser@ajain22-Inspiron-5521:~$
```

Stripes

K Means:

Unsupervised machine learning has a common usage is to find clusters of consumers with common behaviors. In clustering methods, K-means is the most basic and also efficient one. K-Means clustering involve the following logical steps:

- 1) Determine the value of K
 - 2) Determine the initial k centroids
 - 3) Repeat until converge
- Determine membership: Assign each point to the closest centroid
 - Update centroid position: Compute new centroid position from assigned members

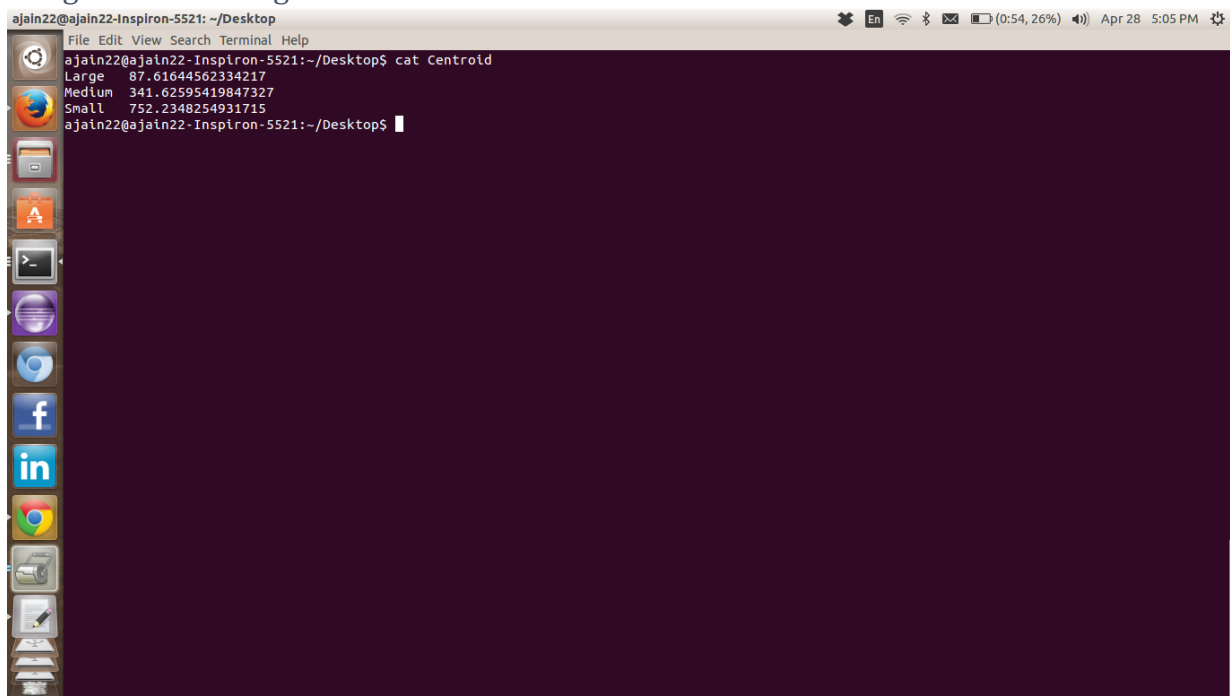
In this problem, we have considered inputs a set of n 1-dimensional points and desired clusters of size 3. Once the k initial centers are chosen, the distance is calculated (Euclidean distance) from every point in the set to each of the 3 centers & point with the corresponding center is emitted by the mapper. Reducer collect all of the points of a particular centroid and calculate a new centroid and emit.

Termination Condition:

When difference between old and new centroid is less than or equal to 0.1

Algorithm:

1. Mappers read input data, compress original data into smaller data sets- auxiliary clusters.
2. Each mapper creates k initial clusters from auxiliary clusters, which are later sent to reducer.
3. Each reducer merges clusters from each mapper and recomputes centroids of all k clusters.
4. These centroids are now streamed back to the original mappers via a broadcast operation.
5. Each mapper now uses new centroids to reassign its auxiliary clusters to these centroids.
6. The mappers send their local clusters back to the reducer.
7. The reducer will merge the clusters again and recompute the centroids.
8. Procedure repeated until reducer decides to stop resending data to mappers i.e. when the algorithm converges.



```
ajain22@ajain22-Inspiron-5521: ~/Desktop
File Edit View Search Terminal Help
ajain22@ajain22-Inspiron-5521:~/Desktop$ cat Centroid
Large      87.61644562334217
Medium    341.62595419847327
Small     752.2348254931715
ajain22@ajain22-Inspiron-5521:~/Desktop$
```

K Means

SHORTEST PATH:

In this problem we are solving the single-source all pairs shortest path using MapReduce using parallel Dijkstra's Algorithm in an iterative manner. The source node is '1'. The source node is processed first, then the nodes connected to the source node are processed and so on.

Input format :

source<tab>adjacency_list : distance_from_the_source:

Termination Condition:

Check the output value with the previous input if any of the distance is still infinity(125 in our code) the continue else convergence is reached and stop.

Algorithm:

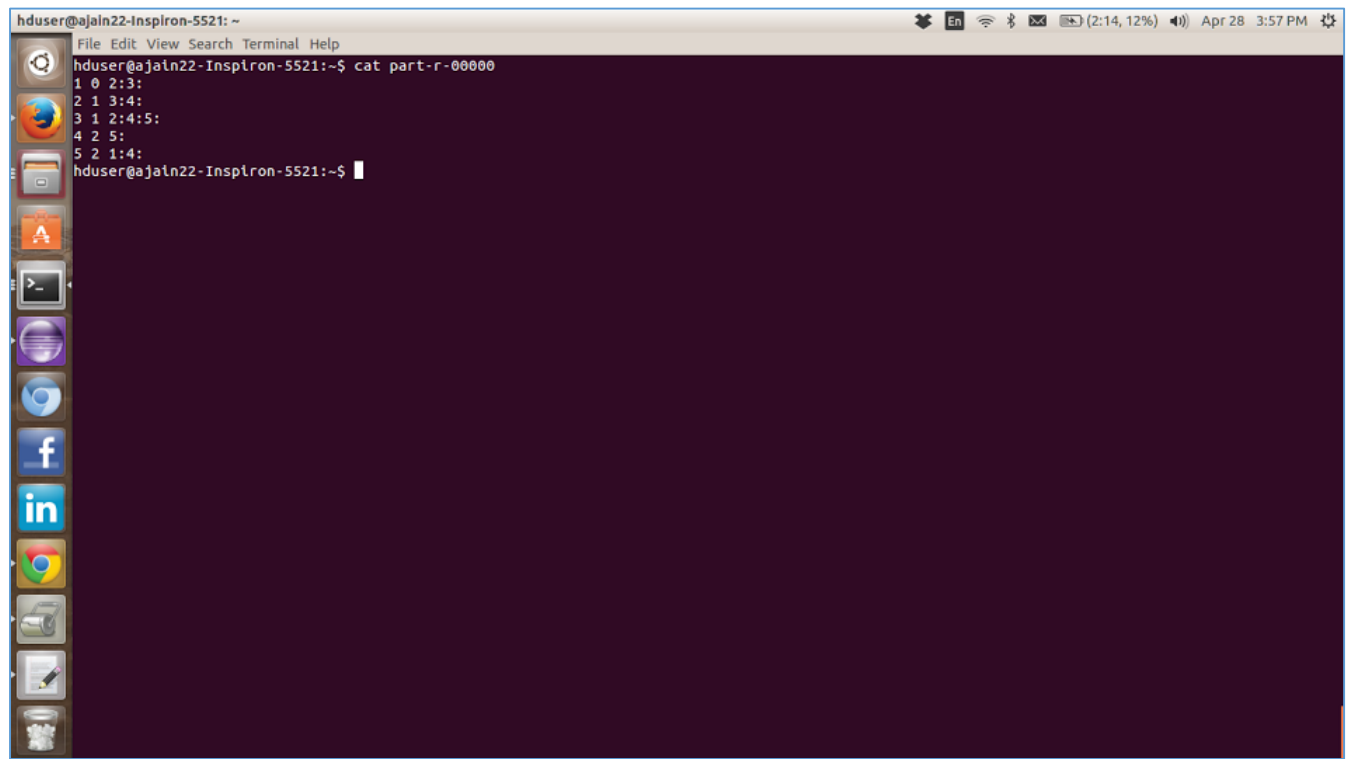
1. Input file having structure as node id and corresponding adjacency list.
2. Mapper reads the file line by line and emit distance of corresponding adjacency list nodes from this node.
3. Reducer find the minimum of the distance of all the distances of a given node.
4. This output file is again read and stored in the map function to check if convergence is reached.

if convergence is reached

stop;

else

repeat step1 with this output file as input file.



```
hduser@ajain22-Inspiron-5521: ~  
File Edit View Search Terminal Help  
hduser@ajain22-Inspiron-5521:~$ cat part-r-00000  
1 0 2:3:  
2 1 3:4:  
3 1 2:4:5:  
4 2 5:  
5 2 1:4:  
hduser@ajain22-Inspiron-5521:~$
```

Shortest Path

Cluster Configuration

Configuration Files

Hadoop configuration is driven by two types of important configuration files:

1. Read-only default configuration - `src/core/core-default.xml`, `src/hdfs/hdfs-default.xml` and `src/mapred/mapred-default.xml`.
2. Site-specific configuration - `conf/core-site.xml`, `conf/hdfs-site.xml` and `conf/mapred-site.xml`.

Environment Variables like `JAVA_HOME` was defined.

`HADOOP_LOG_DIR` - The directory where the daemons' log files are stored. It was set to the default location of the logs at Hadoop Cluster i.e. `/usr/local/Hadoop/logs/usrlogs`

`HADOOP_HEAPSIZE` - The maximum amount of heapsize to use, in MB e.g. 1000MB.

The values for the default configuration and site specific configuration files was also set as provided by the instructor.

Different number of mappers and reducers were employed for different purposes. Like for doing the sorting, two iterations of map reduce jobs were used.

Number of mappers depends on the input file size. Number of reducers were set according to the requirement.

Different Outputs and Visualizations

Outputs have been included above along with the algorithms.

Visualizations are given on the link below:

http://programmaralibi.com/DIC_Report_Project2_Ankit_Milky/#

References

http://hadoop.apache.org/mapreduce/docs/current/mapred_tutorial.html

<http://stackoverflow.com>

<http://codejunkie.net>

<http://puffsun.iteye.com>