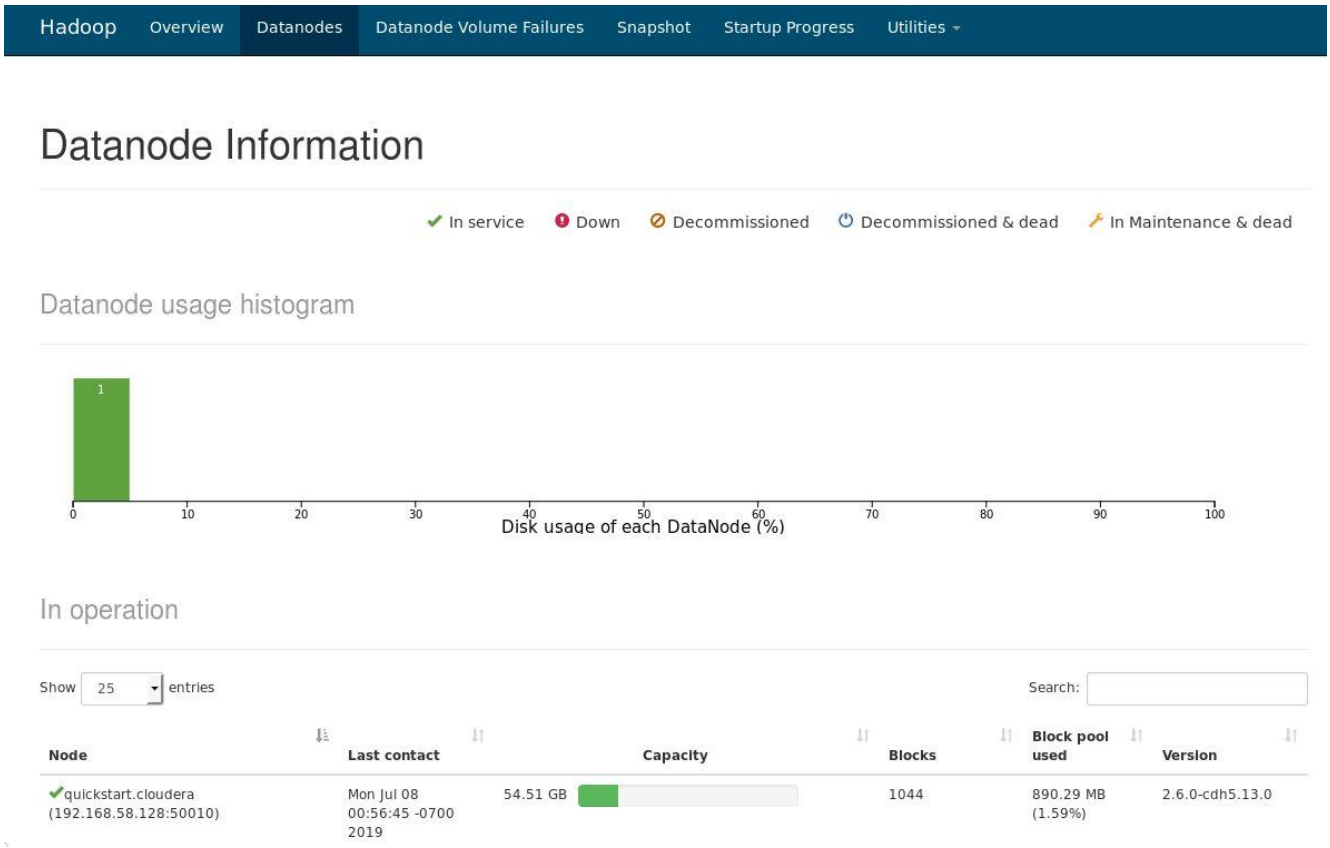
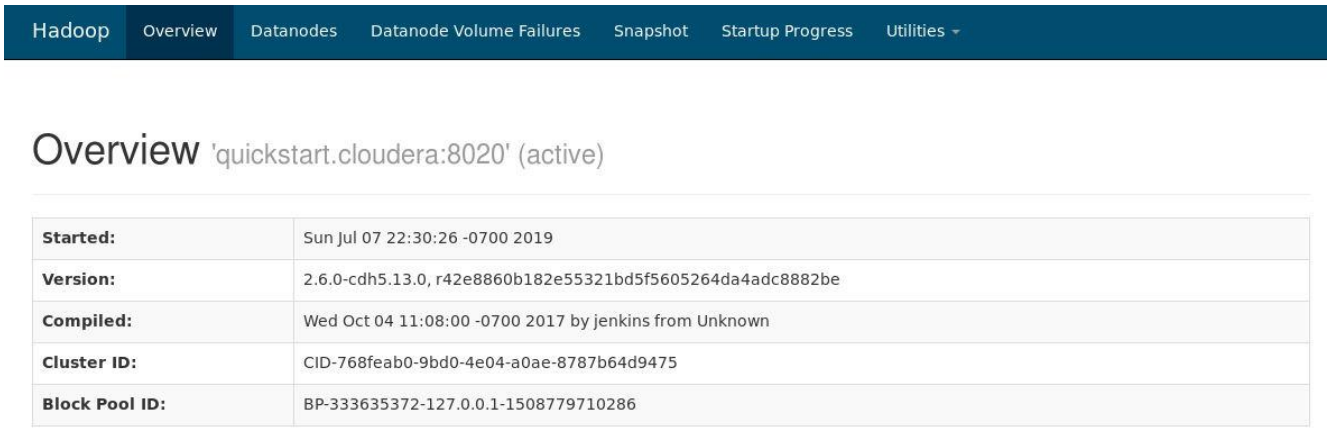
Lab Course - Distributed Data Analytics
Exercise 9

Exercise 1: Preparing the Hadoop infrastructure:

Hadoop infrastructure was successfully set in the local machine and due to some issues with the calling function, I carried out the experiments in a virtual machine, cloudera platform. The results and experiments performed are noted and tabulated.





Summary

Security is off.

Safemode is off.

1,224 files and directories, 1,046 blocks = 2,270 total filesystem object(s).

Heap Memory used 243.66 MB of 348.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 39.47 MB of 39.94 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

Configured Capacity:	54.51 GB
DFS Used:	890.29 MB (1.59%)
Non DFS Used:	18.03 GB
DFS Remaining:	41.85 GB (76.77%)
Block Pool Used:	890.29 MB (1.59%)
DataNodes usages% (Min/Median/Max/stdDev):	1.59% / 1.59% / 1.59% / 0.00%
Live Nodes	1 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Sun Jul 07 22:30:26 -0700 2019
Last Checkpoint Time	Mon Jul 08 00:29:54 -0700 2019

NameNode Journal Status

Current transaction ID: 24156

Journal Manager	State
FileJournalManager(root=/var/lib/hadoop-hdfs/cache/hdfs/dfs/name)	EditLogFileOutputStream(/var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/edits_inprogress_0000000000000024075)

NameNode Storage

Storage Directory	Type	State
/var/lib/hadoop-hdfs/cache/hdfs/dfs/name	IMAGE_AND_EDITS	Active

DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	109.02 GB	1.74 GB (1.59%)	83.45 GB (76.54%)	1.74 GB	2

Hadoop, 2017.

Legacy UI

```
C:\Users\Raaghav>jps
11392 NameNode
7712 NodeManager
11448 Jps
10988 ResourceManager
12716 DataNode
```

Exercise 2 : Analysis of Airport efficiency with Map Reduce

1. Compute the maximum, minimum and average departure delay for each airport.

File name : mapper_mma.py

This file maps the data and sends them as input to the reducer file for finding the maximum, minimum and average departure delay for each airport.

```
import sys

# input comes from STDIN (standard input)

for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    if len(line) >=2:
        dep = line[3]
        delay = line[6]

        print ('%s\t%s' % (dep, delay))
```

File name : reducer_mma.py

This file gets the data from the mapper as input, splits the data to required tasks and finds the maximum, minimum and average delays of departure for each airport.

```
#Reducer.py
import sys

dep_delay = {}

for line in sys.stdin:
    line = line.strip()
    split = line.split('\t')
    if len(split)>1:
        dep = split[0]
        delay = split[1]
    else:
        dep = split[0]
        delay = 0

    if dep in dep_delay:
        dep_delay[dep].append(float(delay))
    else:
        dep_delay[dep] = []
        dep_delay[dep].append(float(delay))

#Reducer
for dep in dep_delay.keys():
    ave_dep = sum(dep_delay[dep])*1.0 / len(dep_delay[dep])
    min_dep = min(dep_delay[dep])
    max_dep = max(dep_delay[dep])
    print ('%s\t%s\t%s\t%s'% (dep, ave_dep,min_dep,max_dep))
```

Procedure:

1. Initially, in the mapper_mma.py, reads the data and splits each line as per the delimiter. Then, we just consider the columns that are related to our exercise. In this case, it is departure airport and the departure delay.

2. Now, the departure airport and the delay are used as input for the reducer. The data is stored in such a way that, the departure airport is used as key and the delays are appended for the airport in the value of the dict.

3. Once a dictionary is created with the key and value pairs, the values appended within each key is used and reduced to a single value as per the requirement.

4. For finding the average departure delay of each airport, the values appended are summed up and averaged to find the same. Also, for the maximum and minimum average delays, min and max functions are used.

Output:

```
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/wc_out_mma/part*
"OTZ"   6.40983606557   -26.0   154.0
"MKG"   1.08620689655   -20.0   202.0
"DAB"   11.6026490066   -19.0   462.0
"MSY"   8.47050482133   -25.0   879.0
"ACT"   12.9519230769   -17.0   202.0
"ONT"   12.5565371025   -21.0   352.0
"CLL"   9.91666666667   -20.0   418.0
"FAT"   12.0459259259   -30.0   601.0
"DSM"   10.4956140351   -18.0   705.0
"MEM"   10.1266149871   -21.0   950.0
"VPS"   22.6535433071   -14.0   1244.0
"PSP"   12.983  -22.0   386.0
"MFE"   7.23076923077   -17.0   327.0
"BHM"   13.7456140351   -18.0   1068.0
"TYR"   6.375   -7.0    58.0
"ISN"   1.47619047619   -20.0   296.0
"AMA"   4.41532258065   -15.0   298.0
"BRW"   -2.725  -29.0   160.0
"LSE"   -7.36363636364  -16.0   2.0
"CLE"   12.0007665772   -25.0   1024.0
"GCK"   6.90322580645   -25.0   452.0
"GSP"   11.9203539823   -16.0   824.0
"OTH"   24.8888888889   -14.0   260.0
"HOU"   10.1724444444   -15.0   365.0
"BET"   -2.48192771084  -27.0   72.0
"WRG"   5.22580645161   -37.0   305.0
"PIB"   4.41509433962   -15.0   282.0
"ADK"   0.666666666667  -34.0   41.0
"XNA"   18.2326139089   -20.0   840.0
"BFL"   -2.51685393258  -20.0   290.0
"RIC"   17.3078125      -20.0   1043.0
"PBG"   6.54838709677   -20.0   164.0
"TRI"   13.5705521472   -14.0   415.0
"MLB"   16.6754385965   -8.0    958.0
"BRO"   7.39622641509   -19.0   428.0
"GSO"   14.0407608696   -14.0   379.0
"MCO"   11.7327155447   -24.0   972.0
"JFK"   12.8124677336   -19.0   1301.0
"ELM"   26.7857142857   -11.0   411.0
"CID"   15.75   -21.0   602.0
"BOS"   9.02041032149   -28.0   1545.0
"GFK"   8.21782178218   -13.0   195.0
"STX"   1.13636363636   -27.0   239.0
"ILM"   6.23902439024   -17.0   389.0
"CLT"   5.36033240997   -23.0   1013.0
"CRP"   3.65829145729   -17.0   348.0
"BIS"   10.4074074074   -18.0   189.0
"SUN"   13.4852941176   -18.0   350.0
"ORD"   13.5450431264   -24.0   1087.0
```

2. Compute a ranking list that contains top 10 airports by their average Arrival delay.

File name : mapper_rank.py

This file maps the data and sends them as input to the reducer file for finding the list of top 10 airports by their average Arrival delay.

```
import sys

# input comes from STDIN (standard input)

for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    if len(line) >=2:
        arr = line[4]
        delay = line[8]

        print ('%s\t%s' % (arr,delay))
```

File name : reducer_rank.py

This file gets the data from the mapper as input, splits the data to required tasks and finds the list of top 10 airports by their average arrival delay.

```
#Reducer.py
import sys
import operator
from collections import OrderedDict

arr_delay = {}


#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('\t')
    if len(split)>1:
        arr = split[0]
        delay = split[1]
    else:
        arr = split[0]
        delay = 0

    if arr in arr_delay:
        arr_delay[arr].append(float(delay))
    else:
        arr_delay[arr] = []
        arr_delay[arr].append(float(delay))
#Reducer
for arr in arr_delay.keys():
    ave_arr = sum(arr_delay[arr])*1.0 / len(arr_delay[arr])
    arr_delay[arr] = ave_arr
arr_delay = OrderedDict(sorted(arr_delay.items(),
                key=operator.itemgetter(1),reverse=True))
delay_rank = {}
print("\n%s\t%s\t%s"%("Arr","Arr_Delay","Rank"))
for rank,arr in enumerate(arr_delay.keys()):
    if rank < 10:
        delay_rank[arr] = arr_delay[arr],rank
        print("%s\t%s\t\t%s"%(arr,arr_delay[arr],rank+1))
```

Procedure:

1. Initially, in the mapper_rank.py, reads the data and splits each line as per the delimiter. Then, we just consider the columns that are related to our exercise. In this case, it is arrival airport and the arrival delay.

2. Now, the arrival airport and the delay are used as input for the reducer. The data is stored in such a way that, the arrival airport is used as key and the delays are appended for the airport in the value of the dict.

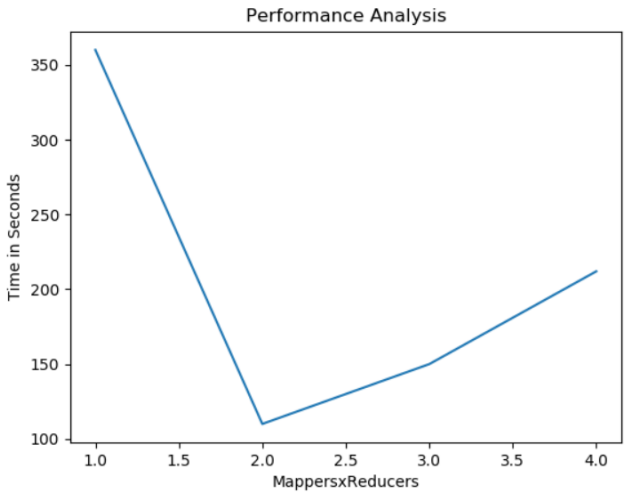
3. Once a dictionary is created with the key and value pairs, the values appended within each key is used and reduced to a single value as per the requirement.

4. For finding the average departure delay of each airport, the values appended are summed up and averaged to find the same.

5. Once the average arrival delays of each airport are found, they are sorted as per the delays and a list of top 10 delays is found from that.

Output – Ranking list of top 10 airports by their average delay:

Arr	Arr_Delay	Rank
"ELM"	81.77	1
"BPT"	47.86	2
"GGG"	46.38	3
"BMI"	37.58	4
"ABI"	34.14	5
"LAW"	29.73	6
"LWS"	29.06	7
"GRB"	24.98	8
"CHA"	24.64	9
"ACT"	24.04	10



Exercise 3: Analysis of Movie dataset using Map and Reduce:

File name : mapper_rating.py

This file maps the data and sends them as input to the reducer file for finding the list of top 10 airports by their average Arrival delay.

```
import sys
import operator
from collections import OrderedDict

mov_dat = {}

dat = []
#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('::')
    if len(split)==3:
        print("%s\t%s\t%s"%(split[0],
                split[1],split[2]))
    else:
        print("%s\t%s\t%s\t%s"%(split[0],
            split[1],split[2],split[3]))
```

File name : reducer_rating.py

This file gets the data from the mapper as input, splits the data to required tasks and finds the list of top 10 airports by their average arrival delay.

```
#Reducer.py
import sys
import operator
from collections import OrderedDict

mov_dat = {}

dat = {}
#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('\t')
    if len(split)==3:
        mid = split[0]
        tit = split[1]
        genre = split[2]
        mov_dat[mid]=tit,str(genre)
    elif len(split)==4:
        gen= mov_dat[split[1]][0]
        rating = split[2]
        if gen in dat:
            dat[gen].append(float(rating))
        else:
            dat[gen] = []
            dat[gen].append(float(rating))
```

```
gen_avg = {}
#Reducer:
for gen in dat.keys():
    avg_rat = sum(dat[gen])*1.0 / len(dat[gen])
    gen_avg[gen] = avg_rat

gen_avg = OrderedDict(sorted(gen_avg.items(),
        key=operator.itemgetter(1),reverse=True))
print("%s\t%s"%("Genre","AvgRating"))
for gen in gen_avg:
    if gen_avg[gen] == 5.0:
        print("%s\t%s"%(gen,gen_avg[gen]))
```

Procedure:

1. The procedure here for the mapper varies a bit because of the usage of 2 input files. Initially, in the mapper_rating.py, reads the data and splits each line as per the delimiter. For ratings data, the size is 4 and hence, the input data is print accordingly and also, for the movies data, the same procedure is carried out.

2. Now, the data are used as input for the reducer. The data is split into a dictionary with movie details and another with movie id and their corresponding ratings

3. Once the dictionaries are created with the key and value pairs, the values of ratings appended within each key is used and reduced to a single value as per the requirement.

4. For finding the average ratings of each movie id, the values appended are summed up and averaged to find the same.

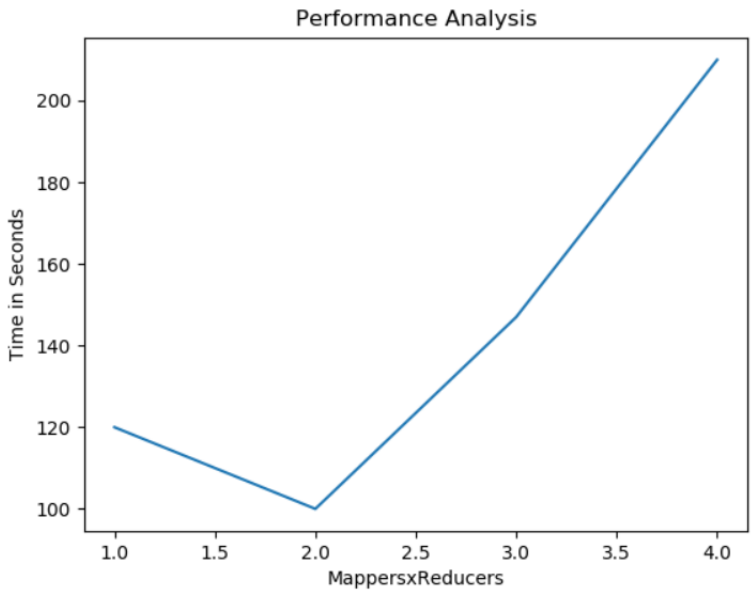
5. Once the average ratings of each movie are found, they are sorted as per the delays and a list of top movies with ratings equal to 5.0 is listed with their name being retrieved from the movie details dictionary.

10M dataset

```
Avg.Rat Movie Name
5.0     Shadows of Forgotten Ancestors (1964)
5.0     Fighting Elegy (Kenka erejii) (1966)
5.0     Blue Light, The (Das Blaue Licht) (1932)
5.0     Sun Alley (Sonnenallee) (1999)
5.0     Satan's Tango (Sátántangó) (1994)
```

1M dataset

```
File  Edit  View  Search  Terminal  Help
Avg.Rat Movie Name
5.0     Lured (1947)
5.0     Gate of Heavenly Peace, The (1995)
5.0     Baby, The (1973)
5.0     Smashing Time (1967)
5.0     One Little Indian (1973)
5.0     Bittersweet Motel (2000)
5.0     Ulysses (Ulisse) (1954)
5.0     Song of Freedom (1936)
5.0     Schlafes Bruder (Brother of Sleep) (1995)
5.0     Follow the Bitch (1998)
```



```
File  Edit  View  Search  Terminal  Help
^C[cloudera@quickstart ~]$  hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/ha
op-streaming-2.6.0-mr1-cdh5.13.0.jar -Dmapred.map.tasks=1 -Dmapred.reduce.tasks=1 -file /h
ome/cloudera/mapper_user.py /home/cloudera/reducer_user.py -mapper "python mapper_user.py"
 -reducer "python reducer_user.py" -input /user/ratings.dat -output /user/cloudera/out_080
7_r2
19/07/08 04:00:47 WARN streaming.StreamJob: -file option is deprecated, please use generic
 option -files instead.
packageJobJar: [/home/cloudera/mapper_user.py, /home/cloudera/reducer_user.py] [/usr/lib/h
adoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob3870175309004163901.ja
r tmpDir=null
19/07/08 04:01:00 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/07/08 04:01:01 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/07/08 04:01:05 INFO mapred.FileInputFormat: Total input paths to process : 1
19/07/08 04:01:06 INFO mapreduce.JobSubmitter: number of splits:2
19/07/08 04:01:06 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead,
 use mapreduce.job.maps
19/07/08 04:01:06 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Inste
ad, use mapreduce.job.reduces
19/07/08 04:01:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_156256391195
5_0026
19/07/08 04:01:10 INFO impl.YarnClientImpl: Submitted application application_156256391195
5_0026
19/07/08 04:01:10 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera
:8088/proxy/application_1562563911955_0026/
19/07/08 04:01:10 INFO mapreduce.Job: Running job: job_1562563911955_0026
19/07/08 04:02:42 INFO mapreduce.Job: Job job_1562563911955_0026 running in uber mode : fa
lse
19/07/08 04:02:42 INFO mapreduce.Job:  map 0% reduce 0%
19/07/08 04:03:19 INFO mapreduce.Job:  map 3% reduce 0%
19/07/08 04:03:20 INFO mapreduce.Job:  map 6% reduce 0%
19/07/08 04:03:25 INFO mapreduce.Job:  map 8% reduce 0%
19/07/08 04:03:26 INFO mapreduce.Job:  map 10% reduce 0%
19/07/08 04:03:32 INFO mapreduce.Job:  map 15% reduce 0%
19/07/08 04:03:38 INFO mapreduce.Job:  map 20% reduce 0%
19/07/08 04:03:45 INFO mapreduce.Job:  map 22% reduce 0%
19/07/08 04:03:46 INFO mapreduce.Job:  map 24% reduce 0%
19/07/08 04:03:52 INFO mapreduce.Job:  map 27% reduce 0%
19/07/08 04:03:58 INFO mapreduce.Job:  map 32% reduce 0%
19/07/08 04:04:04 INFO mapreduce.Job:  map 36% reduce 0%
19/07/08 04:04:10 INFO mapreduce.Job:  map 42% reduce 0%
19/07/08 04:04:16 INFO mapreduce.Job:  map 45% reduce 0%
19/07/08 04:04:22 INFO mapreduce.Job:  map 46% reduce 0%
19/07/08 04:04:23 INFO mapreduce.Job:  map 49% reduce 0%
19/07/08 04:04:29 INFO mapreduce.Job:  map 51% reduce 0%
19/07/08 04:04:36 INFO mapreduce.Job:  map 52% reduce 0%
19/07/08 04:04:48 INFO mapreduce.Job:  map 54% reduce 0%
19/07/08 04:04:54 INFO mapreduce.Job:  map 57% reduce 0%
19/07/08 04:05:00 INFO mapreduce.Job:  map 59% reduce 0%
19/07/08 04:05:01 INFO mapreduce.Job:  map 61% reduce 0%
19/07/08 04:05:07 INFO mapreduce.Job:  map 64% reduce 0%
```

2. Find the user who has assigned lowest average rating among all users who rated more than 40

File name : mapper_user.py

This file maps the data and sends them as input to the reducer file for finding the list of top 10 airports by their average Arrival delay.

```
import sys

# input comes from STDIN (standard input)

for line in sys.stdin:
    line = line.strip()
    line = line.split("::")

    if len(line) >=2:
        movieID = line[0]
        rating = line[2]

        print ('%s\t%s' % (movieID,float(rating)))
```

File name : reducer_user.py

This file gets the data from the mapper as input, splits the data to required tasks and finds the list of top 10 airports by their average arrival delay.

```
import sys
import operator
from collections import OrderedDict

arr_delay = {}
rat = {}

#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('\t')
    if len(split)>1:
        arr = split[0]
        delay = split[1]
    else:
        arr = split[0]
        delay = 0

    if arr in arr_delay:
        arr_delay[arr].append(1)
        rat[arr].append(float(delay))
    else:
        arr_delay[arr] = []
        rat[arr] = []
        arr_delay[arr].append(1)
        rat[arr].append(float(delay))

#Reducer
user_rating={}
for arr in arr_delay.keys():
    if  sum(arr_delay[arr]) > 40:
        user_rating[arr] = sum(rat[arr])*1.0 / len(arr_delay[arr])
user_rating = OrderedDict(sorted(user_rating.items(),
                    key=operator.itemgetter(1),reverse=False))

for i in user_rating:
    print(i,user_rating[i])
    break
```

Procedure:

1. Initially, in the mapper_user.py, reads the data and splits each line as per the delimiter. Then, we just consider the columns that are related to our exercise. In this case, it is user id and rating by the users.

2. Now, the user id and ratings are used as input for the reducer. The data is stored in such a way that, the user id is used as key and the ratings are appended for the users in the value of the dict.

3. Once a dictionary is created with the key and value pairs, the values appended within each key is used and reduced to a single value as per the requirement.

4. For finding the average ratings of each users who rated more than 40 times, first the users are selected whose length of total ratings is greater than 40. Now with user id as key, the values appended are summed up and averaged to find the same.

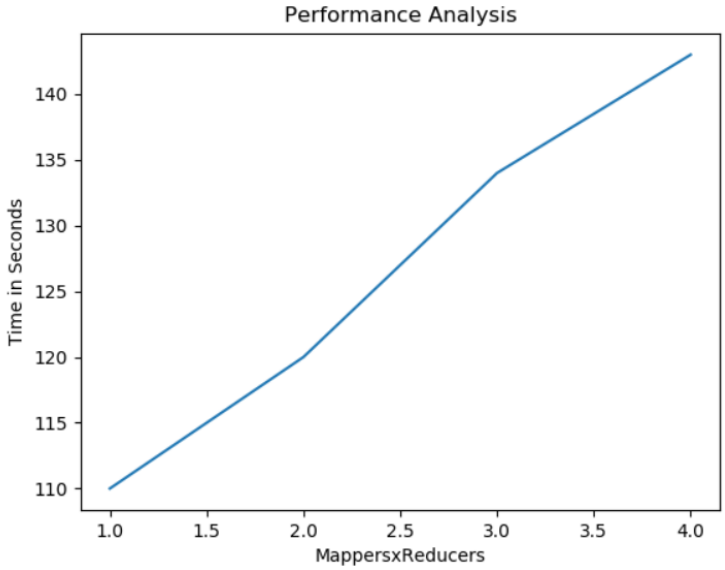
5. After averaging, the values are sorted in ascending order and the user is found.

1M dataset

```
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/wc_out_user_1/part*
('3598', 1.0153846153846153)
```

10M dataset

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/out_0807_r2/part*
('24176', 1.0)
```



1. Find the highest average rated movie genre :

File name : mapper_genre.py

This file maps the data and sends them as input to the reducer file for finding the list of top 10 airports by their average Arrival delay.

```
import sys
import operator
from collections import OrderedDict

mov_dat = {}

dat = []
#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('::')
    if len(split)==3:
        print("%s\t%s\t%s"%(split[0],
                split[1],split[2]))
    else:
        print("%s\t%s\t%s\t%s"%(split[0],
            split[1],split[2],split[3]))
```

File name : reducer_genre.py

This file gets the data from the mapper as input, splits the data to required tasks and finds the list of top 10 airports by their average arrival delay.

```
#Reducer.py
import sys
import operator
from collections import OrderedDict

mov_dat = {}

dat = {}
#Partitoner
for line in sys.stdin:
    line = line.strip()
    split = line.split('\t')
    if len(split)==3:
        mid = split[0]
        tit = split[1]
        genre = split[2]
        mov_dat[mid]=tit,str(genre)
    elif len(split)==4:
        gen= mov_dat[split[1]][0]
        rating = split[2]
        if gen in dat:
            dat[gen].append(float(rating))
        else:
            dat[gen] = []
            dat[gen].append(float(rating))
```

```
gen_avg = {}
#Reducer:
for gen in dat.keys():
    avg_rat = sum(dat[gen])*1.0 / len(dat[gen])
    gen_avg[gen] = avg_rat

gen_avg = OrderedDict(sorted(gen_avg.items(),
        key=operator.itemgetter(1),reverse=True))
print("%s\t%s"%("Genre","AvgRating"))
for gen in gen_avg:
    print("%s\t%s"%(gen,gen_avg[gen]))
    break
```

Procedure:

1. The procedure here for the mapper varies a bit because of the usage of 2 input files. Initially, in the mapper_rating.py, reads the data and splits each line as per the delimiter. For ratings data, the size is 4 and hence, the input data is print accordingly and also, for the movies data, the same procedure is carried out.

2. Now, the data are used as input for the reducer. The data is split into a dictionary with movie details and another with movie id and their corresponding ratings

3. Once the dictionaries are created with the key and value pairs, the values of ratings appended within each key is used and reduced to a single value as per the requirement.

4. For finding the average ratings of each movie id, the values appended are summed up and averaged to find the same.

5. Once the average ratings of each movie are found, they are sorted as per the delays and a list of top movies with ratings equal to 5.0 is listed with their genre being retrieved from the movie details dictionary.

10M dataset

```
Genre                   AvgRating
Animation|IMAX|Sci-Fi   4.75
```

1M dataset

```
File  Edit  View  Search  Terminal  Help
Genre                       AvgRating
Animation|Comedy|Thriller       4.4738372093
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
```