Lab Course: Distributed Data Analytics
Exercise Sheet 5
Group 2 - Monday

Submitted by: Sruthy Annie Santhosh, 312213

Topic:Distributed Machine Learning (Supervised)

- My System

  Hardware and Software SetUp:

Processor	M1 chip
Operating System	MacOS Monterey
Number of Cores	8
RAM	8 GB
Python version	3.8.8

  The code editor used in this sheet : VSCode and Jupyter Notebook (graph)
  The output images attached are screenshots from the terminal.

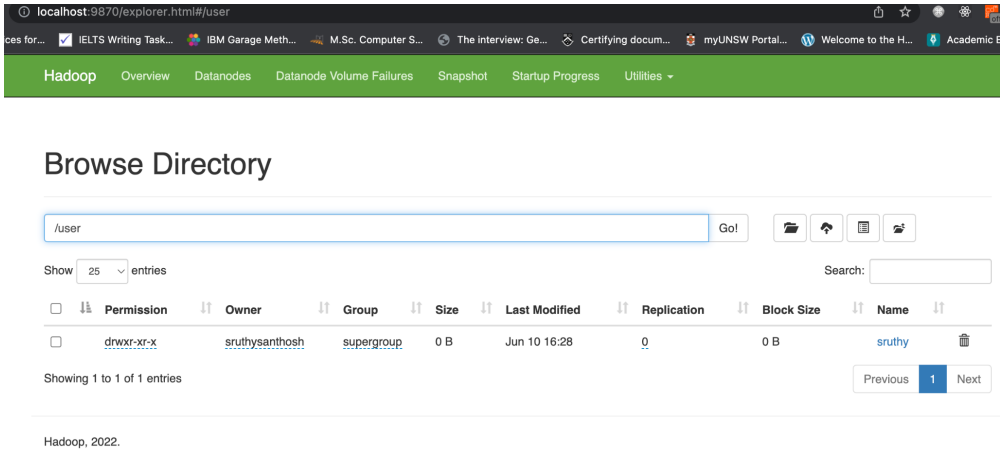
- Exercise 1: Preparing your Hadoop infrastructure

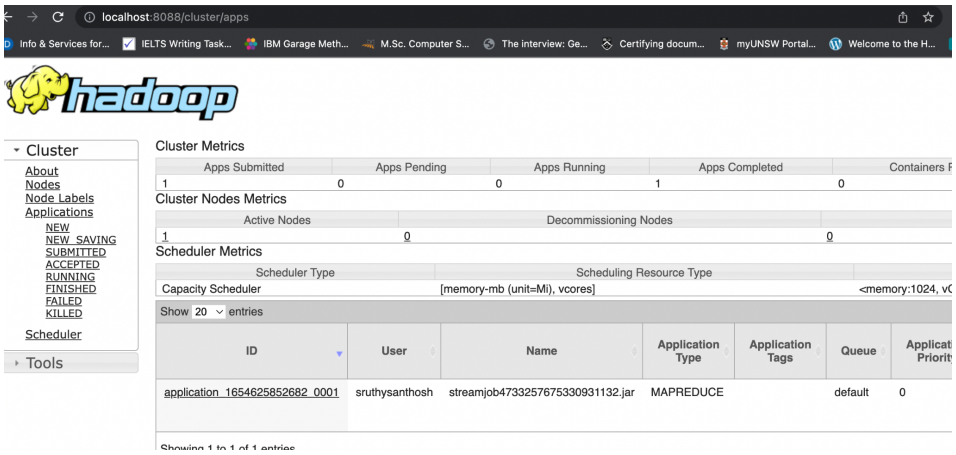
  I have set up the Pseudo Distributed mode of Hadoop in my local system. Testing of the correct implementation was done by creating directories and putting files in the directories. The output of jps is as follows:

```
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % jps
1062 SecondaryNameNode
1257 ResourceManager
21722 Jps
1660 NameNode
1357 NodeManager
925 DataNode
(base) sruthysanthosh@Sruthys-MacBook-Air ~ %
```

```
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hadoop version
Hadoop 3.3.3
Source code repository https://github.com/apache/hadoop.git -r d37586cbda38c338d9fe481addda5a05fb516f71
Compiled by stevel on 2022-05-09T16:36Z
Compiled with protoc 3.7.1
From source with checksum eb96dd4a797b6989ae0cdb9db6efc6
This command was run using /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/common/hadoop-common-3.3.3.jar
(base) sruthysanthosh@Sruthys-MacBook-Air ~ %
```

Hadoop user-interface was also familiarised with:





Word Count MapReduce Example

I have run the word count example using python mapper and reducer files. One mapper and one reducer were used. The given text file was first transferred to the hadoop directory using the -put command.

```
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as sruthysanthosh in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
localhost: datanode is running as process 925.  Stop it first and ensure /tmp/hadoop-sruthysanthosh-datanode.pid file is empty before retry.
Starting secondary namenodes [Sruthys-MacBook-Air.local]
Sruthys-MacBook-Air.local: secondarynamenode is running as process 1062.  Stop it first and ensure /tmp/hadoop-sruthysanthosh-secondarynamenode.pid file i
2022-06-07 20:18:27,754 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting resourcemanager
resourcemanager is running as process 1257.  Stop it first and ensure /tmp/hadoop-sruthysanthosh-resourcemanager.pid file is empty before retry.
Starting nodemanagers
localhost: nodemanager is running as process 1357.  Stop it first and ensure /tmp/hadoop-sruthysanthosh-nodemanager.pid file is empty before retry.
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % jps
1062 SecondaryNameNode
2217 Jps
1257 ResourceManager
1660 NameNode
1357 NodeManager
925 DataNode
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % cd Downloads
(base) sruthysanthosh@Sruthys-MacBook-Air Downloads % python version
python: can't open file 'version': [Errno 2] No such file or directory
(base) sruthysanthosh@Sruthys-MacBook-Air Downloads % python --version
Python 3.8.8
(base) sruthysanthosh@Sruthys-MacBook-Air Downloads % cd
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hdfs dfs -mkdir /user
2022-06-07 20:28:39,077 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hdfs dfs -mkdir /user/sruthy
2022-06-07 20:29:05,743 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hdfs dfs -put /Users/sruthysanthosh/Downloads/text.txt /user/sruthy
```

In the mapper file

The lines from the input file are read and preprocessed to get each word.
Then each word is sent as the output of the mapper along with the value 1.
Mapper output- (word,1),(word,1).....

```python
import sys

#Reading each line in the input file
for line in sys.stdin:
    line = line.strip() #Removing white space characters from beginning and end
    words = line.split() #Splitting based on space
    for word in words:
        #Sending each word and value 1 to reducer
        print('%s\t%s' % (word, 1))
```

Reducer file

In the reducer, mapper outputs after being sorted according to the key. Hence all same words would be together.
The counting of each word is done here.

```
# read the entire line from STDIN, which is the output of the mapper
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # splitting based on tab character as we wrote in mapper file
    word, count = line.split('\t'),1
    # convert count to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, discard
        continue

    # Hadoop sorts the map output by keys , ie, word before passing to reducer
    # Hence all same words will be together
    if current_word == word:
        #If the current word is same as new word, add count
        current_count += count
    else:
        if current_word:
            #end of one word, hence writing its count
            # write result to STDOUT
            #Current word and its count is overwrittern after each word is over
            print('%s\t%s' % (current_word, current_count))
        #Take the new word as current words, and its count as current count
        current_count = count
        current_word = word
```

Output of Reducer is each word and its count

The command used for doing mapreduce is : The output is stored in a file in the hadoop directory names output. The paths of mapper and reducer are specified along with the file names.

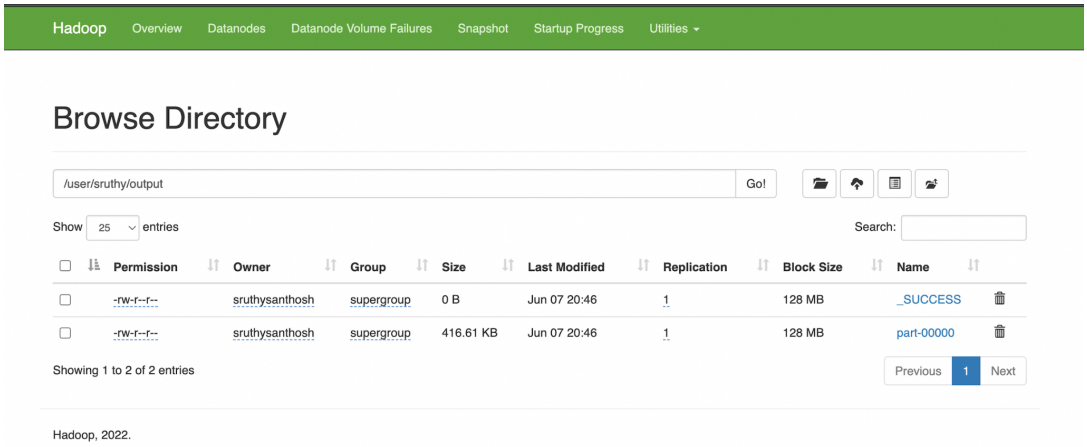
```
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools/lib/hadoop-streaming-3.3.3.jar -input /user/sruthy/text.txt
-output /user/sruthy/output -mapper mapper1.py -reducer reducer1.py -file /Users/sruthysanthosh/Downloads/mapper1.py -file /Users/sruthysanthosh/Downloads/reducer1.py
```

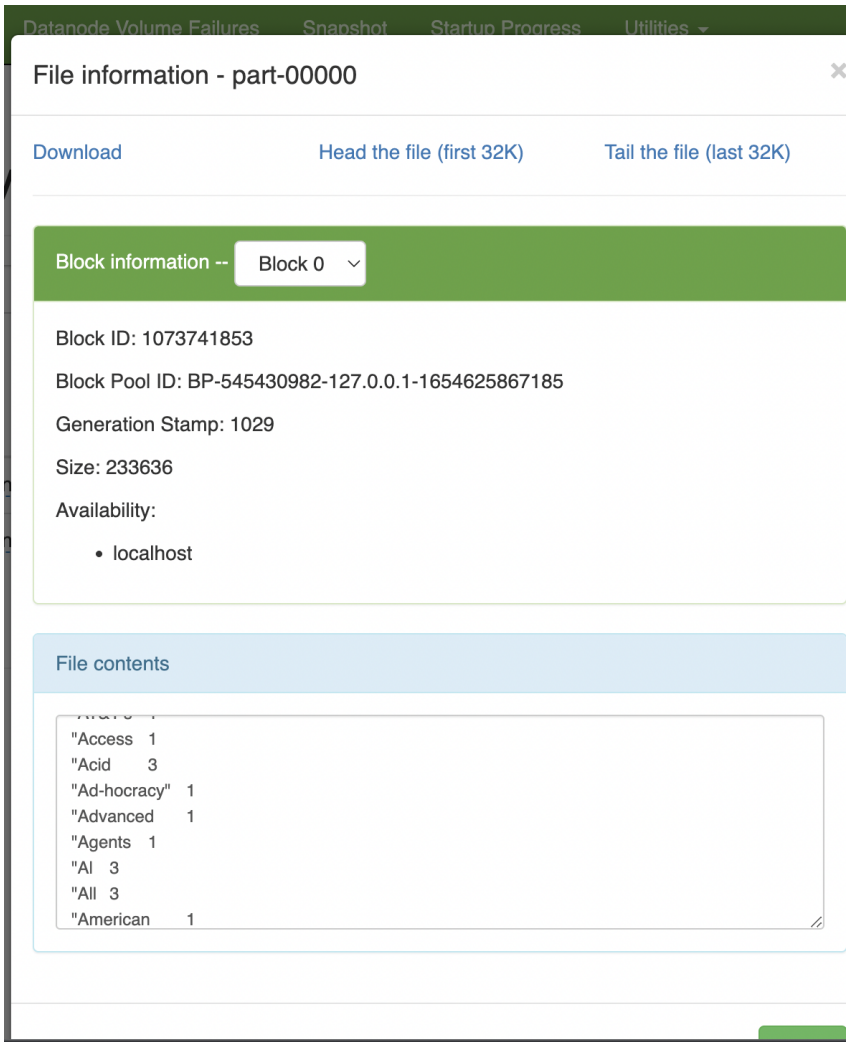
The console output is as follows:

```
(base) sruthysanthosh@Sruthys-MacBook-Air ~ % hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools/lib/hadoop-streaming-3.3.3.jar -input /user/sruthy/text.txt
-output /user/sruthy/output -mapper mapper1.py -reducer reducer1.py -file /Users/sruthysanthosh/Downloads/mapper1.py -file /Users/sruthysanthosh/Downloads/reducer1.py


2022-06-07 20:45:55,214 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
2022-06-07 20:45:55,502 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
packageJobJar: [/Users/sruthysanthosh/Downloads/mapper1.py, /Users/sruthysanthosh/Downloads/reducer1.py] [] /var/folders/9z/75yp7_695hg75lbf0tpmjs9w0000gn/T/streamjob162735347403
1150597.jar tmpDir=null
2022-06-07 20:45:56,680 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2022-06-07 20:45:56,826 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2022-06-07 20:45:56,826 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2022-06-07 20:45:56,857 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2022-06-07 20:45:57,704 INFO mapred.FileInputFormat: Total input files to process : 1
2022-06-07 20:45:57,851 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-07 20:45:58,128 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1343776106_0001
2022-06-07 20:45:58,129 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-06-07 20:45:58,518 INFO mapred.LocalDistributedCacheManager: Localized file:/Users/sruthysanthosh/Downloads/mapper1.py as file:/tmp/hadoop-sruthysanthosh/mapred/local/job_lo
cal1343776106_0001_953118fd-f1dd-46a4-a99e-1ec55d66e5d6/mapper1.py
2022-06-07 20:45:58,557 INFO mapred.LocalDistributedCacheManager: Localized file:/Users/sruthysanthosh/Downloads/reducer1.py as file:/tmp/hadoop-sruthysanthosh/mapred/local/job_l
ocal1343776106_0001_be9a1570-5664-46c8-b061-909c96b29f9a/reducer1.py
2022-06-07 20:45:58,776 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2022-06-07 20:45:58,781 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2022-06-07 20:45:58,783 INFO mapreduce.Job: Running job: job_local1343776106_0001
2022-06-07 20:45:58,786 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
2022-06-07 20:45:58,796 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-06-07 20:45:58,796 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-06-07 20:45:58,862 INFO mapred.LocalJobRunner: Waiting for map tasks
2022-06-07 20:45:58,864 INFO mapred.LocalJobRunner: Starting task: attempt_local1343776106_0001_m_000000_0
2022-06-07 20:45:58,894 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-06-07 20:45:58,894 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-06-07 20:45:58,906 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on Linux.
2022-06-07 20:45:58,906 INFO mapred.Task:  Using ResourceCalculatorProcessTree : null
2022-06-07 20:45:58,936 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/user/sruthy/text.txt:0+665057
2022-06-07 20:45:58,994 INFO mapred.MapTask: numReduceTasks: 1
2022-06-07 20:45:59,022 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2022-06-07 20:45:59,023 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2022-06-07 20:45:59,023 INFO mapred.MapTask: soft limit at 83886080
2022-06-07 20:45:59,023 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2022-06-07 20:45:59,023 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2022-06-07 20:45:59,027 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2022-06-07 20:45:59,044 INFO streaming.PipeMapRed: PipeMapRed exec [/Users/sruthysanthosh/./mapper1.py]
2022-06-07 20:45:59,160 INFO Configuration.deprecation: mapred.work.output.dir is deprecated. Instead, use mapreduce.task.output.dir
2022-06-07 20:45:59,165 INFO Configuration.deprecation: mapred.local.dir is deprecated. Instead, use mapreduce.cluster.local.dir
2022-06-07 20:45:59,166 INFO Configuration.deprecation: map.input.file is deprecated. Instead, use mapreduce.map.input.file
2022-06-07 20:45:59,166 INFO Configuration.deprecation: map.input.length is deprecated. Instead, use mapreduce.map.input.length
2022-06-07 20:45:59,166 INFO Configuration.deprecation: mapred.job.id is deprecated. Instead, use mapreduce.job.id
```

```
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=10024
2022-06-10 11:16:26,093 INFO mapred.LocalJobRunner: Finishing task: attempt_local2880116
2022-06-10 11:16:26,094 INFO mapred.LocalJobRunner: reduce task executor complete.
2022-06-10 11:16:26,369 INFO mapreduce.Job:  map 100% reduce 100%
2022-06-10 11:16:26,370 INFO mapreduce.Job: Job job_local288011680_0001 completed succes
2022-06-10 11:16:26,392 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=10659488
                FILE: Number of bytes written=17300709
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=54129196
                HDFS: Number of bytes written=10024
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=450018
                Map output records=450018
                Map output bytes=4427247
                Map output materialized bytes=5327289
                Input split bytes=108
                Combine input records=0
                Combine output records=0
                Reduce input groups=299
                Reduce shuffle bytes=5327289
                Reduce input records=450018
                Reduce output records=298
                Spilled Records=900036
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=107
                Total committed heap usage (bytes)=1068498944
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=27064598
        File Output Format Counters
                Bytes Written=10024
2022-06-10 11:16:26,392 INFO streaming.StreamJob: Output directory: /user/sruthy/output1
```

The output file in hadoop





- Exercise 2: Analysis of Airport efficiency with Map Reduce

The required csv file was downloaded containing 450017 lines. The file was transferred to the hadoop directory using the -put command. I have used one mapper and one reducer for implementing each of the tasks.

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

Mapper file:

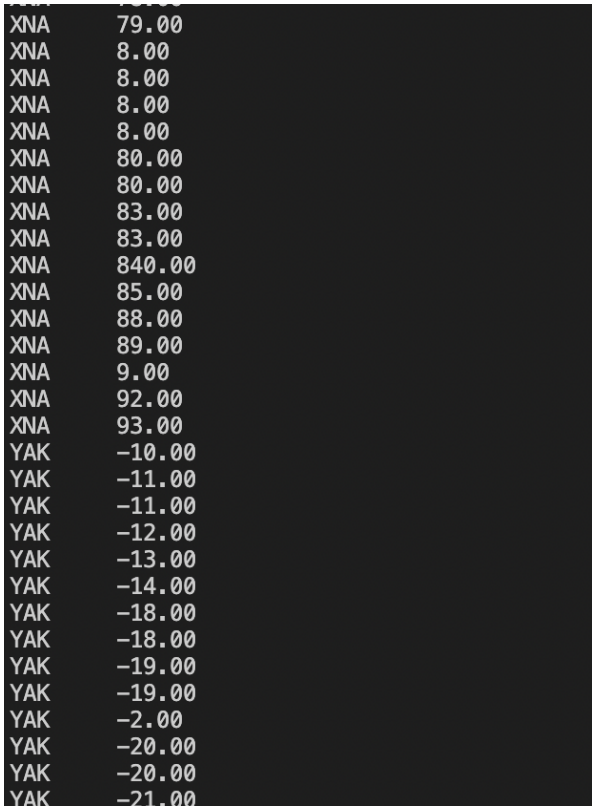
```
import sys

#For each line in input
for line in sys.stdin:

    # Removing leading, trailing whitespace and quotes
    line = line.strip().replace('\"', '')
    # getting each attribute by splitting using , as it is a csv file
    line = line.split(",")

    if len(line) >= 2:
        #Getting the required column values
        dep_airport = line[3]
        dep_delay = line[6]

        #If column value is empty, assigning 0
        if dep_delay == '':
            dep_delay = 0

        #Forming the output of mapper using tab character
        string = '%s\t%s' % (dep_airport, dep_delay)
        print(string)
```

In the mapper, we get each attribute after splitting. Then the required attributes which are the departure airport and departure delay are selected and send as output, separated by a tab character.

The sample output of mapper:

```
PHL     -10.00
CLT     -8.00
RDU     -7.00
CLT     -7.00
PHL     -3.00
FLL     -7.00
PHL     -5.00
RDU     -8.00
PHX     -4.00
ORD     -6.00
IND     -13.00
ALB     -14.00
CLT     -7.00
BWI     -6.00
CLT     28.00
MCO     -7.00
PHL     -8.00
CLT     -7.00
PIT     -9.00
CLT     -5.00
PHL     13.00
RSW     11.00
CLT     4.00
PBI     23.00
PHL     -10.00
ORD     -5.00
PHL     -1.00
PHX     -7.00
PHL     -8.00
PHX     -16.00
```

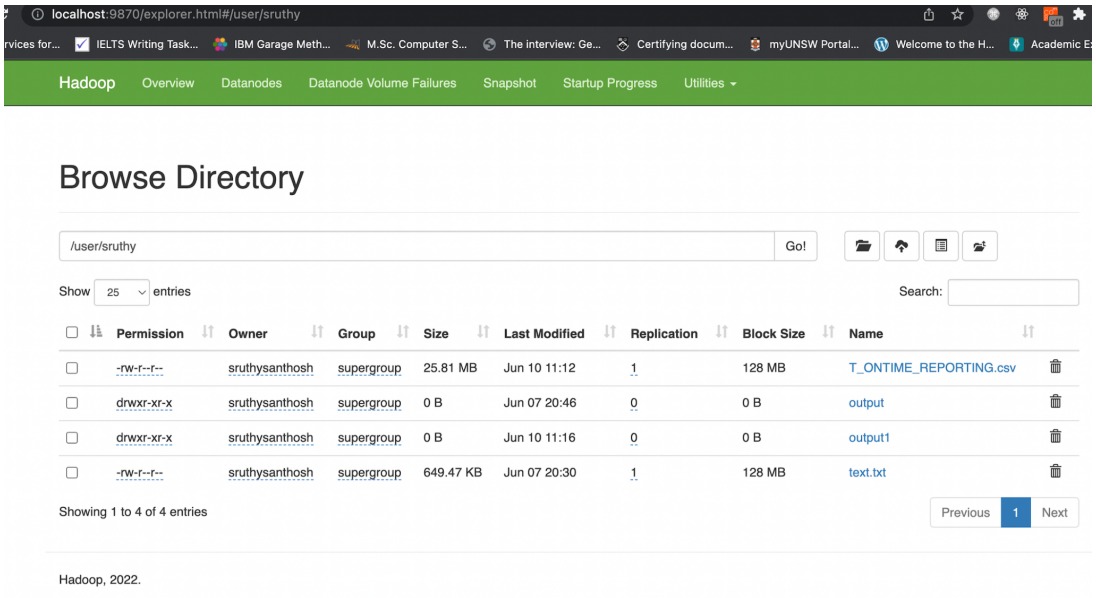
The different airports will be sorted accordingly and send as input to reducer.
Sample input of reducer:

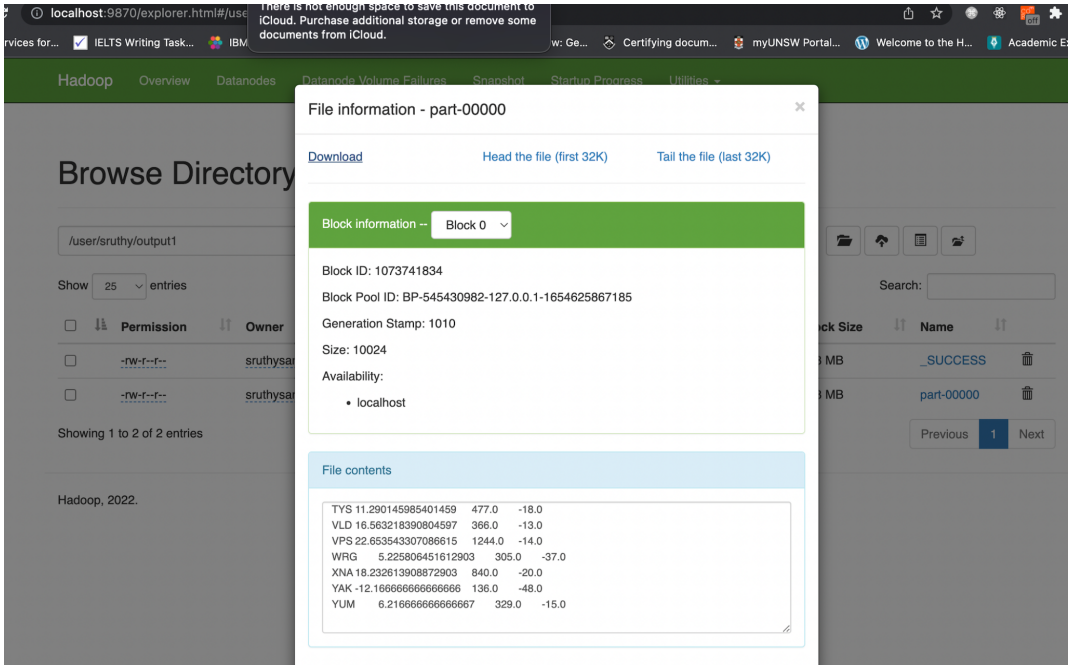


Reducer File:

```
#for each line in the input of reducer
# ( mapper output after sorting based on key)
for line in sys.stdin:
    #Removing trailing white spaces
    line = line.strip()
    #Getting the key and values send by mapper by splitting
    #according to tab character
    dep_airport, dep_delay= line.split('\t')
    #print(line)
    #Converting string to float as we need the dep delay in float
    try:
        dep_delay = float(dep_delay)
    except ValueError:
        continue

    #Forming dictionary for each unique key, with values as list
    if dep_airport in dict:
        dict[dep_airport].append(float(dep_delay))
    else:
        dict[dep_airport] = []
        dict[dep_airport].append(float(dep_delay))

#For each departure airport
for dep_airport in dict.keys():
    #each value in dict is a list of delays for that airport
    #average delay calc
    avg_delay = sum(dict[dep_airport])*1.0 / len(dict[dep_airport])
    #max delay
    max_delay = max(dict[dep_airport])
    #min delay
    min_delay = min(dict[dep_airport])

    #Forming output of reducer by using tab as separator
    #Format - Departure Airport, Average delay, Maximum delay, Minimum delay
    string = '%s\t%s\t%s\t%s' % (dep_airport, avg_delay,max_delay,min_delay)
    #printing to output
    print(string)
```

Each key and value received from mapper is appended to a dictionary. Then the calculations are done on the dictionary. Each key in dictionary corresponds to a departure airport and its value will be a list of all the delays of that airport.
Average, Maximum and minimum delays are found for each airport as shown in code.
The output of the Mapreduce is stored in the output file in Hadoop.





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

Mapper File: It is similar to the mapper file for the above question. Here we take the arrival airport and arrival delay attributes and then pass them as the output. The format of the output is also same.

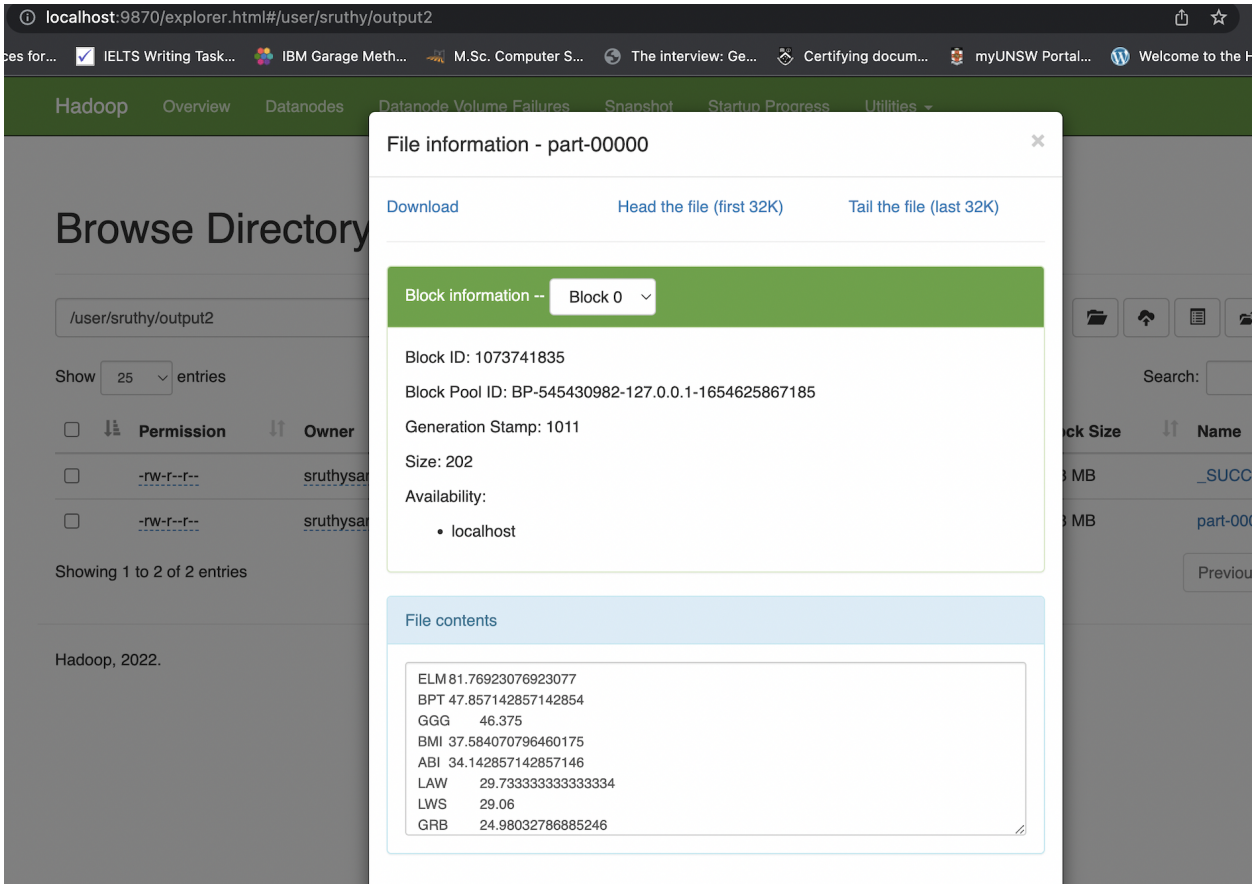
```
#For each line in input
for line in sys.stdin:

    # Removing leading, trailing whitespace and quotes
    line = line.strip().replace('\"', '')
    # getting each attribute by splitting using , as it is a csv file
    line = line.split(",")

    if len(line) >= 2:
        #Getting the required column values
        arr_airport = line[4]
        arr_delay = line[8]

        #If column value is empty, assigning 0
        if arr_delay == '':
            arr_delay = 0

        #Forming the output of mapper using tab character
        string = '%s\t%s' % (arr_airport, arr_delay)
        print(string)
```

Reducer File: In Reducer, like the previous question, the keys and values are taken and stored as dictionary. Hence each key would be an arrival airport and its value would be a list of delays corresponding to that airport. Then the average delay for each key is calculated and this is stored in another dictionary. This dictionary is then sorted in descending order, to get the airports having highest average rating. The output is written to Hadoop directory

```
#for each line in the input of reducer
# ( mapper output after sorting based on key)
for line in sys.stdin:
    #Removing trailing white spaces
    line = line.strip()
    #Getting the key and values send by mapper by splitting
    #according to tab character
    arr_airport, arr_delay= line.split('\t')

    #Converting string to float as we need the arr delay in float
    try:
        arr_delay = float(arr_delay)
    except ValueError:
        continue

    #Forming dictionary for each unique key, with values as list
    if arr_airport in dict:
        dict[arr_airport].append(float(arr_delay))
    else:
        dict[arr_airport] = []
        dict[arr_airport].append(float(arr_delay))

#For each arrival airport
for arr_airport in dict.keys():
    #each value in dict is a list of delays for that airport
    #average delay calc
    avg_delay = sum(dict[arr_airport])*1.0 / len(dict[arr_airport])

    #adding the average delay of each airport to a new dictionary
    new_dict[arr_airport]= avg_delay
```

```
count = 10

#Sorting in descending order based on the values
sorted_keys = sorted(new_dict, key=new_dict.get, reverse=True)

#Getting the 10 airports having highest average delay
for i in sorted_keys:
    #Forming output
    string = '%s\t%s' %(i, new_dict[i])
    print(string)

    count = count-1
    if(count==0):
        break
```

The output: The top 10 airports are ELM, BPT, GGG, BMI, ABI, LAW, LWS ,GRB, CHA and ACT.



The output files of all questions are attached.

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

Two files were downloaded for this question: movies.dat and ratings.dat. These files were merged on the Movie ID column to get the file merge_data5.csv. These files were then transferred to the hadoop directory using the -put command. In this exercise, time taken by each mapper X reducer also needs to be found, hence I have added time also to the hadoop streaming command. The number of reducers and mappers can be set using -D mapred.job.maps=x -D mapred.job.reduces=x .

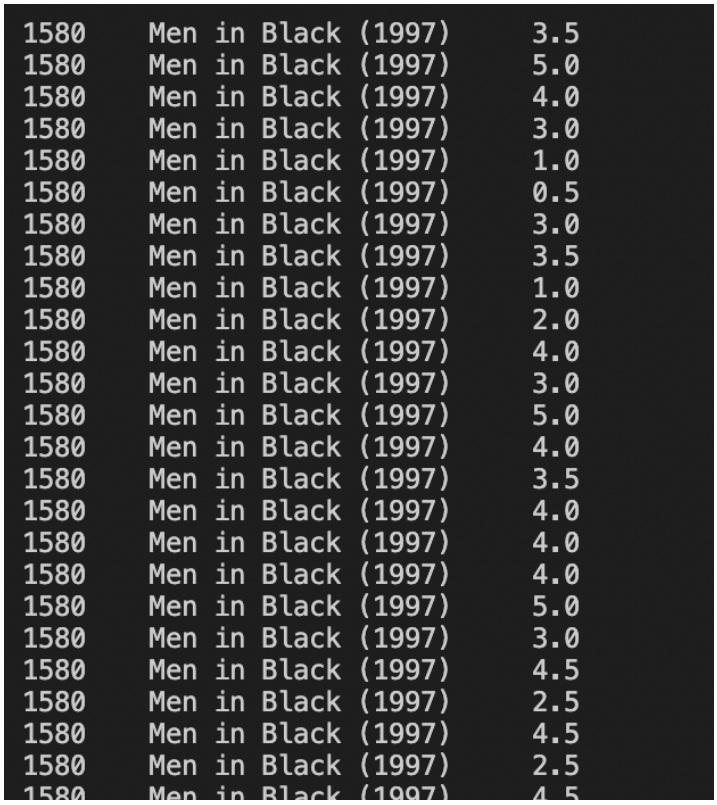
1. Find the movie title which has the maximum average rating?

Mapper file: Here I have used the merged file for finding the movie title having the maximum average rating. First each line is read and then

split accordingly. Then the required attributes are send as output of the mapper. Here I have collected the movie id, movie title and rating and then send them separated by a tab character. Since I used merge function on movieId, the file would be already sorted based on movie Id.

```
import sys
#For each line in input
for line in sys.stdin:

    # Removing leading, trailing whitespace and quotes
    line = line.strip().replace('\"', '')
    # getting each attribute by splitting using , as it is a csv file
    line = line.split(",")

    if len(line) >= 2:
        #Getting the required column values
        movie_id = line[0]
        movie_name = line[1]
        rating = line[4]

        #Forming the output of mapper using tab character
        string = '%s\t%s\t%s' % (movie_id, movie_name,rating)
        print(string)
```

Sample Output of mapper file:



Reducer File: In the reducer file, the values send as output by mapper are collected. A dictionary is created with key as movie name and value as a list of ratings given for that movie. Then the average rating for each key is found and appended to a new dictionary.

```
#for each line in the input of reducer
# ( mapper output after sorting based on key)
for line in sys.stdin:
    #Removing trailing white spaces
    line = line.strip()
    #Getting the key and values send by mapper by splitting
    #according to tab character
    movie_id, movie_name, rating= line.split('\t')

    #Converting string to float as we need the rating in float
    try:
        rating= float(rating)
    except ValueError:
        continue

    #Forming dictionary for each unique key, with values as list
    if movie_name in dict:
        dict[movie_name].append(float(rating))
    else:
        dict[movie_name] = []
        dict[movie_name].append(float(rating))

#For each movie name
for name in dict.keys():
    #each movie name is a key and its values contain a list of all
    # the ratings it got
    #Average rating calculation
    avg_rating = sum(dict[name])*1.0 / len(dict[name])

    #New dictionary with key as movie name and average rating as va
    new_dict[name]= avg_rating
```

Then the new dictionary is sorted in descending order and we see that the highest average rating is 5 and 5 movies have that. These are written separated by tab to the output file in Hadoop directory.

```
max = 5.0

#Sorting the new dictionary is descending order
sorted_keys = sorted(new_dict, key=new_dict.get, reverse=True)

#for each movie name, if average rating is equal to the maximum rating,
#printing the same
for i in sorted_keys:
    if(new_dict[i] == max):
        #Forming output of the movie names with highest average rating
        string = '%s\t%s' %(i, new_dict[i])
        print(string)
```

Terminal:

```
                Merged Map outputs=8
                GC time elapsed (ms)=930
                Total committed heap usage (bytes)=6759645184
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=700272367
        File Output Format Counters
                Bytes Written=204
022-06-10 15:25:04,248 INFO streaming.StreamJob: Output directory: /user/sruthy/movieoutput1
adoop jar  -input /user/sruthy/merge_data5.csv -output  -mapper  -reducer     75.68s user 8.07s system 139% cpu 1:00.18 total
base) sruthysanthosh@Sruthys-MacBook-Air ~ %
```

Hadoop Interface:

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

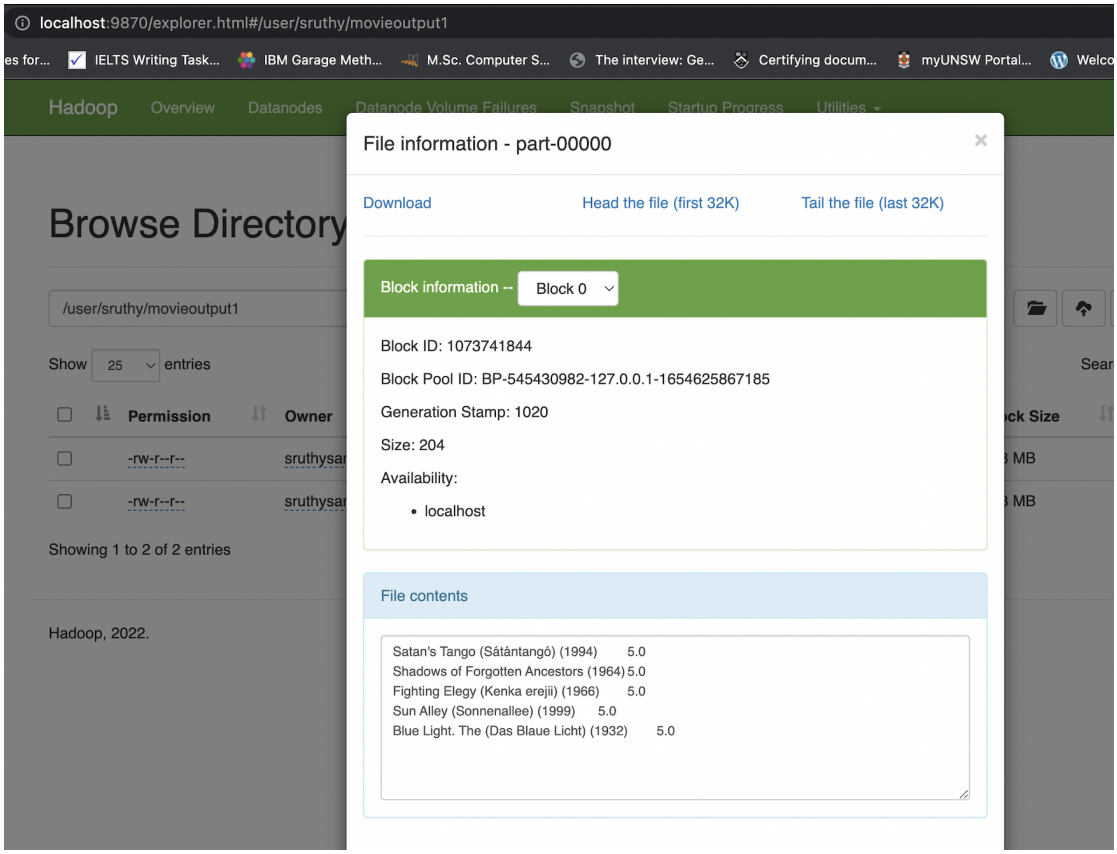
/user/sruthy Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	sruthysanthosh	supergroup	25.81 MB	Jun 10 11:12	1	128 MB	T_ONTIME_REPORTING.csv
-rw-r--r--	sruthysanthosh	supergroup	667.81 MB	Jun 10 15:21	1	128 MB	merge_data5.csv
drwxr-xr-x	sruthysanthosh	supergroup	0 B	Jun 10 15:25	0	0 B	movieoutput1
drwxr-xr-x	sruthysanthosh	supergroup	0 B	Jun 07 20:46	0	0 B	output
drwxr-xr-x	sruthysanthosh	supergroup	0 B	Jun 10 11:16	0	0 B	output1
drwxr-xr-x	sruthysanthosh	supergroup	0 B	Jun 10 11:22	0	0 B	output2
-rw-r--r--	sruthysanthosh	supergroup	252.82 MB	Jun 10 15:21	1	128 MB	ratings.dat
-rw-r--r--	sruthysanthosh	supergroup	649.47 KB	Jun 07 20:30	1	128 MB	text.txt

Showing 1 to 8 of 8 entries Previous 1 Next

Output:



The movie titles having the maximum average rating of 5 are Satan's Tango, Shadows of Forgotten Ancestors, Fighting Elegy, Sun Alley, Blue Light, The.

I have analysed the time for different number of mappers and reducers.

Number of Mappers	Number of Reducers	Time in seconds
1	1	75.68s
2	2	76.81s
4	4	76.29s

Number of Mappers	Number of Reducers	Time in seconds
2	1	74.55s

4	1	75.15s
8	1	77.53s

Number of Mappers	Number of Reducers	Time in seconds
1	2	76.94s
1	4	76.30s
1	8	74.85s

From the above tables we can see that, on increasing both mappers and reducers, the time taken increases slightly (<1s). This can be due to computation or overhead of having many mappers and reducers.

When increasing only mappers, we see that initially the time decreases and then increases after 4 mappers. Also for increase in only reducers time also decreased slightly. But overall the time changes is very very less when changing the number of mappers and reducers. This can be because for this operation, increasing mappers and reducers may not help much.

But still we can say that increasing reducers help in decreasing time taken. Since reducing function does the main operation, it makes sense to have more reducers.

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

Here I have only used the ratings.dat file as input.

Mapper File: As done in previous question, the mapper splits and gets the required attributes. Here we collect the user Id and the rating. These values are the output of the mapper file.

```
#For each line in input file
for line in sys.stdin:

    # Removing leading, trailing whitespace and quotes
    line = line.strip().replace('\"', '')
    # getting each attribute by splitting using :: as it is a dat file
    line = line.split("::")

    if len(line) >=2 :
        #Getting the required column values
        user_id = line[0]
        rating = line[2]

        #Forming the output of mapper using tab character
        string = '%s\t%s' % (user_id, rating)
        print(string)
```

Then the output of mapper is sorted according to user ids and send as the input to Reducer. Hence all same userIds will be together.

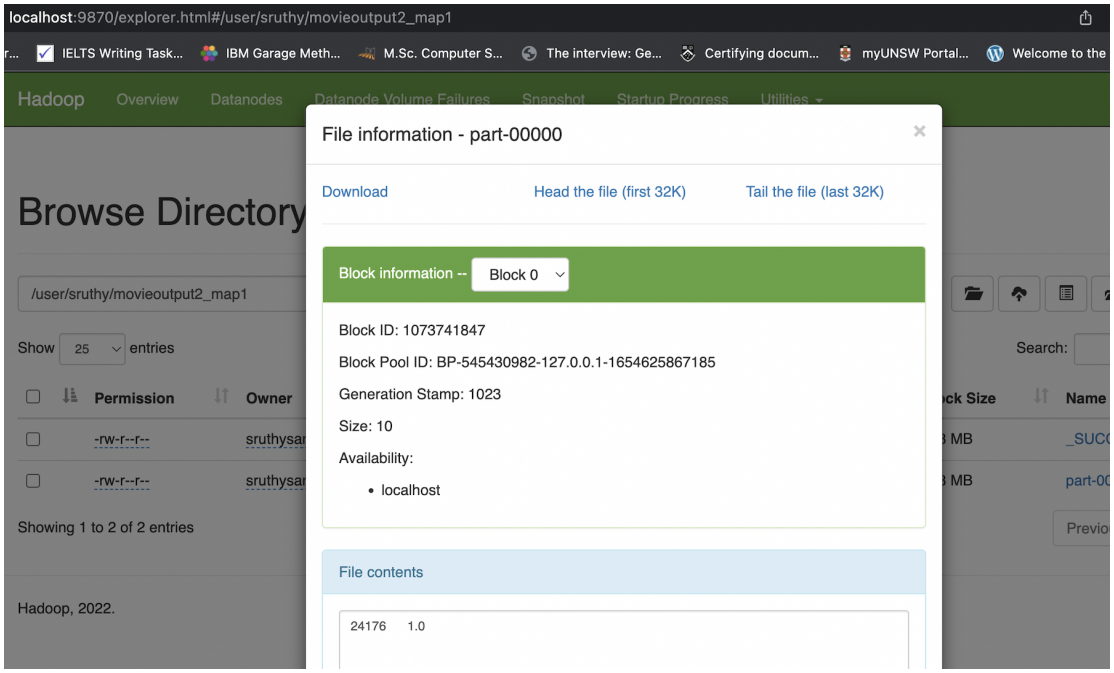
Reducer File: In the reducer file, the inputs are split and then appended to a dictionary. The userIds will be keys and the values will be the list of ratings which were given by that user.
Then the users having more than 40 ratings in their value is selected and the average of the ratings for each such user is calculated and stored in a new dictionary. This dictionary is then sorted in ascending order. The first key in the sorted dictionary will contain the user who assigned the lowest average rating among users who have rated more than 40 times.

```
#for each line in the input of reducer
# ( mapper output after sorting based on key)
for line in sys.stdin:
    #Removing trailing white spaces
    line = line.strip()
    #Getting the key and values send by mapper by splitting
    #according to tab character
    user_id, rating= line.split('\t')

     #Converting string to float as we need the rating in float
    try:
        rating= float(rating)
    except ValueError:
        continue

    #Forming dictionary for each unique key, with values as list
    if user_id in dict:
        dict[user_id].append(float(rating))
    else:
        dict[user_id] = []
        dict[user_id].append(float(rating))

#For each user in dictionary
#each userid is a key and its values contain a list of all
# the ratings given by that user
for user in dict.keys():
    #If the number of ratings given by user is greater than or equal to 40
    if(len(dict[user])>=40):
        #Average rating calculation
        avg_rating = sum(dict[user])*1.0 / len(dict[user])

        #New dictionary with userid as key and the average rating as value
        new_dict[user]= avg_rating
```

```
min = 1.0
#Sorting dictionary in ascending order
sorted_keys = sorted(new_dict, key=new_dict.get)

for i in sorted_keys:
    #Finding the user with the least average rating
    if(new_dict[i] == min):
        #Forming the output as :userid  average rating
        string = '%s\t%s' %(i, new_dict[i])
        print(string)
```

The output of the reducer would be the user Id and the average rating fulfilling the above conditions.

Hadoop Interface:



The output is userId 24178 having an average rating of 1.0

Number of Mappers	Number of Reducers	Time in seconds
1	1	65.23s
2	2	65.53s
4	4	66.31s

Number of Mappers	Number of Reducers	Time in seconds
1	2	64.39s
1	4	64.05s
1	8	65.44s

Number of Mappers	Number of Reducers	Time in seconds
2	1	64.44s
4	1	65.10
8	1	64.60

From the tables we can see that the trend is similar to that of above question. On increasing both mapper and reducer, the time increases very slightly. While increasing only reducers, the time decreases and then increases for 8 reducers. In the case where we increase only mappers, we see that time slightly increases and then decreases. The change is very very slight and can be due to CPU usage at the time.

Thus we can say the overall the time increases with increase in number of mappers and reducers as the computation overhead will be higher.

3. Find the highest average rated movie genre?

Here I used the merged csv file.

Mapper File: As done in previous question, the mapper splits and gets the required attributes. Here we collect the genre and the rating. These values are the output of the mapper file. The values are sorted according to genre before sending as input to reducer.

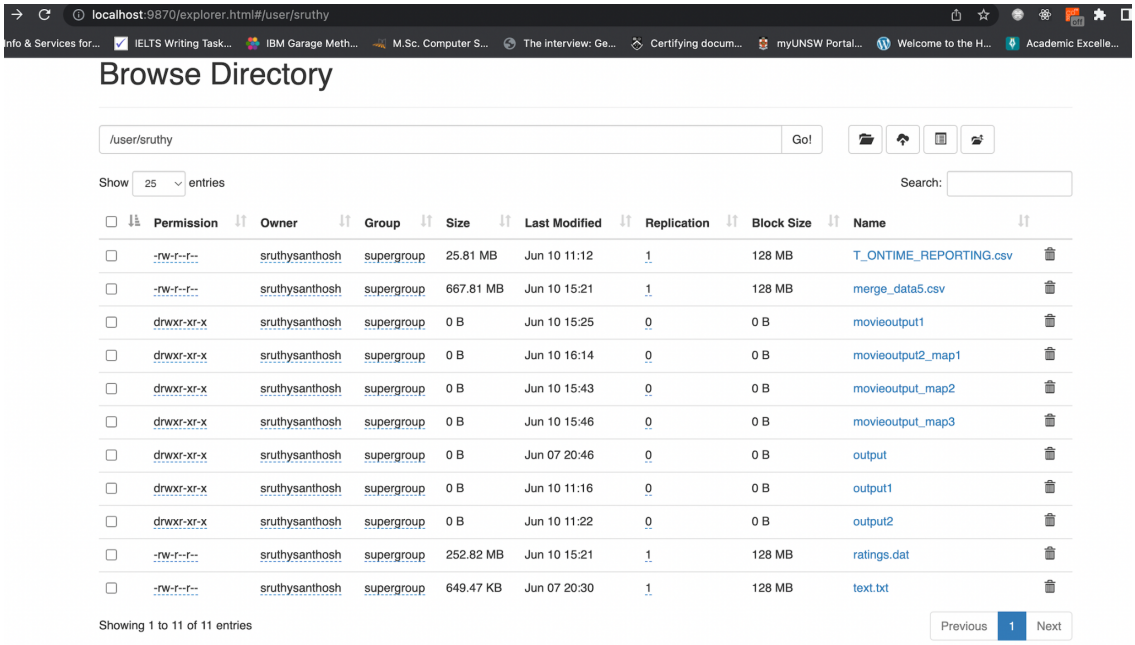
```
#For each line in input file
for line in sys.stdin:

    # Removing leading, trailing whitespace and quotes
    line = line.strip().replace('\"', '')

    # getting each attribute by splitting using , as it is a csv file
    line = line.split(",")

    if len(line) >=2 :
        #Getting the required column values
        genre = line[2]
        rating = line[4]

        #Forming the output of mapper using tab character
        string = '%s\t%s' % (genre, rating)
        print(string)
```

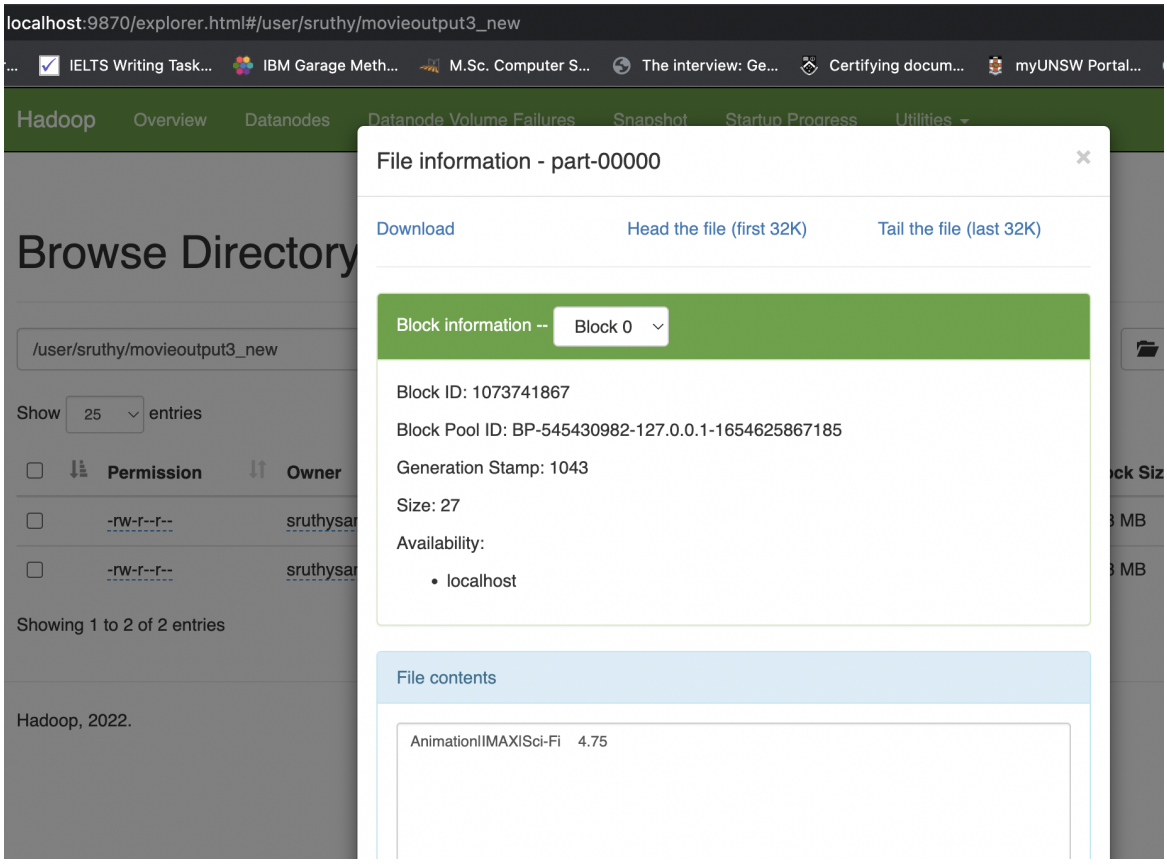
Reducer File: In the reducer file, the inputs are split and then appended to a dictionary. The genres will be keys and the values will be the list of ratings which were given for each combination of genre.
Then the average of the ratings for each such genre list is calculated and stored in a new dictionary. This dictionary is then sorted in descending order. The first key in the sorted dictionary will contain the genre having the highest average rating. The output of reducer would be the genre and the average rating separated by a tab character. This will be written to the output directory mentioned in the hadoop command.

```
#for each line in the input of reducer
# ( mapper output after sorting based on key)
for line in sys.stdin:
    #Removing trailing white spaces
    line = line.strip()
    #Getting the key and values send by mapper by splitting
    #according to tab character
    genre, rating= line.split('\t')

    #Converting string to float as we need the rating in float
    try:
        rating= float(rating)
    except ValueError:
        continue

     #Forming dictionary for each unique key, with values as list
    if genre in dict:
        dict[genre].append(float(rating))
    else:
        dict[genre] = []
        dict[genre].append(float(rating))

#For each genre categories corresponding to movies in dictionary
#each genre list is a key and its values contain a list of all
# the ratings movies belonging to that genre list got
for genre in dict.keys():
    #Average rating calculation
    avg_rating = sum(dict[genre])*1.0 / len(dict[genre])

 #New dictionary with genre listas key and the average rating as value
    new_dict[genre]= avg_rating
```

```
count = 1

##Sorting dictionary in descending order to get the highest average rating
sorted_keys = sorted(new_dict, key=new_dict.get, reverse=True)

for i in sorted_keys:
    if(count>0):
        #The genre list having the highest average rated movies is found
        #Output is formed as : Genre list  Average rating
        string = '%s\t%s' %(i, new_dict[i])
        print(string)
        count = count - 1
```

Hadoop Interface:





The genre having the highest rating of 4.75 is Animation|IMAX|Sci-Fi

Sample Terminal:

Number of Mappers	Number of Reducers	Time in seconds
1	1	60.64s
2	2	62.52s
4	4	60.68s

Number of Mappers	Number of Reducers	Time in seconds
1	2	61.11s
1	4	60.21s
1	8	62.51

Number of Mappers	Number of Reducers	Time in seconds
2	1	61.62s
4	1	60.66s
8	1	60.62s

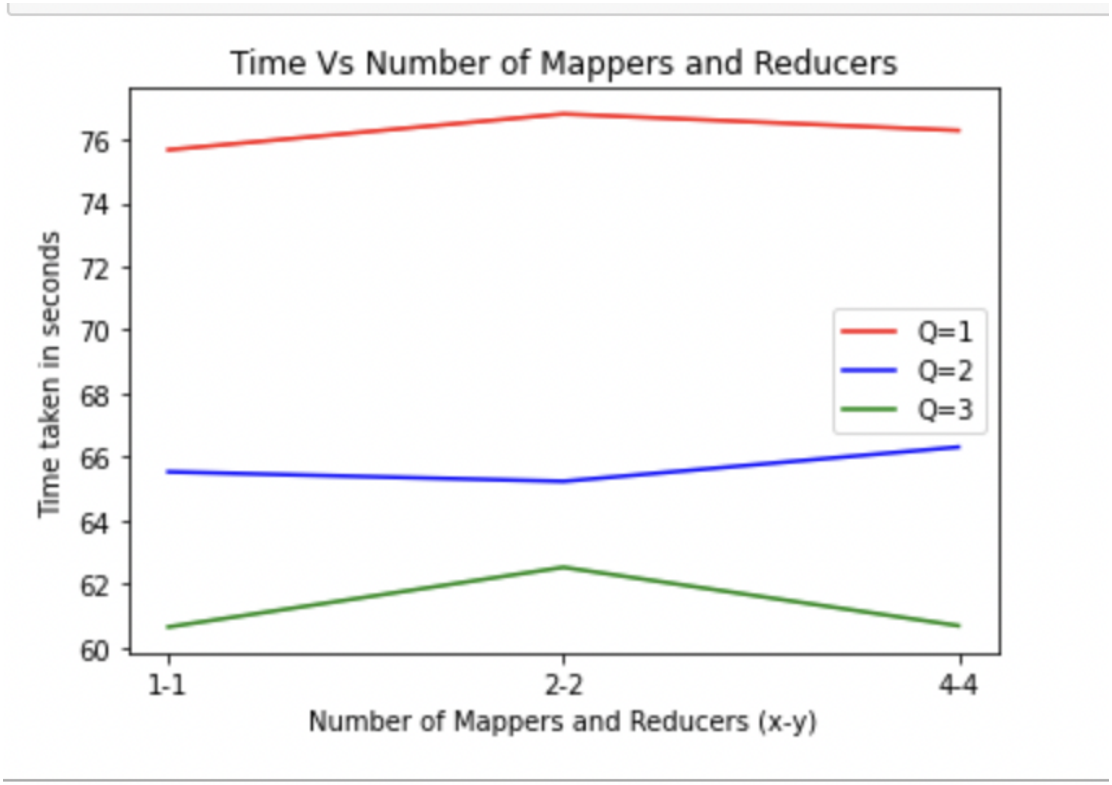
From the tables we can see that here, when we increase both mappers and reducers, the time increases and then decreases. This can be due to randomness and

the CPU usage. While increasing only reducers , we see a slight decrease initially, then increases. When we increase only mappers, we see that there is a slight decrease in time.

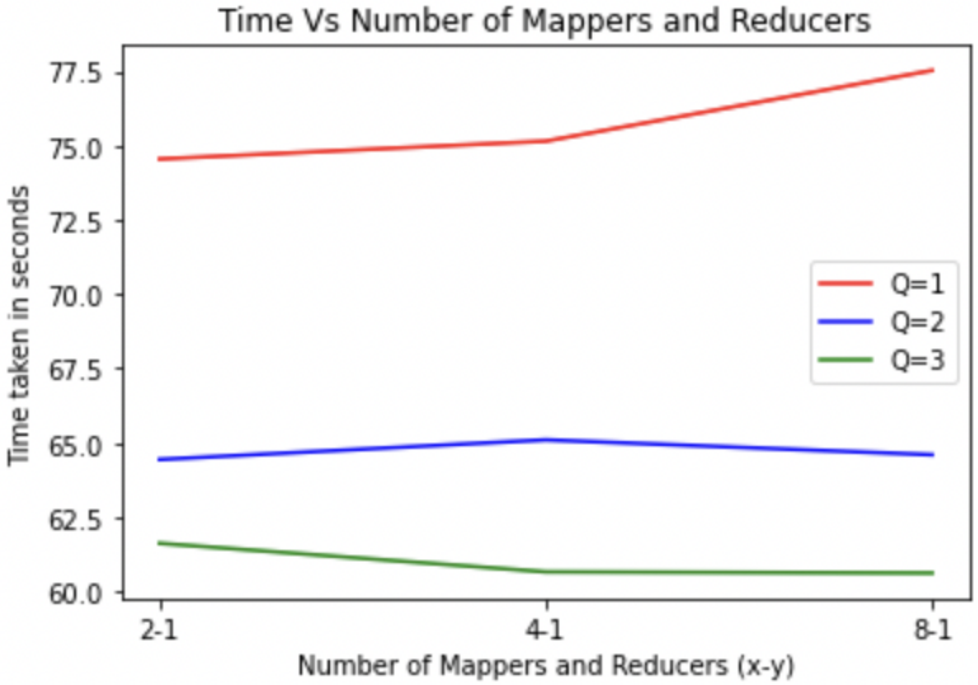
Thus overall we can say that increasing only mappers or reducers show a slightly increase in trend initially. But this is still very very slight and can be considered as randomness also (not sure).

I have written the code for merging of datasets and the code for plotting graphs in Jupyternotebook which is also attached.

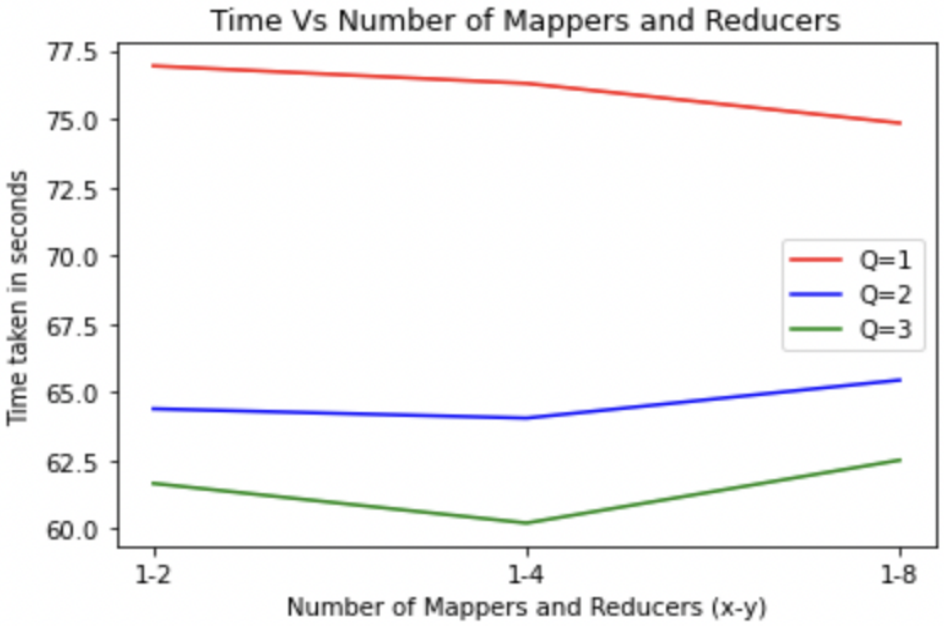
Graphs



We see that for Question 1 and 3, time decreases after initially increasing



We see that time varies very slightly.



We can see that the time decreases initially and then increases for Question 2 and 3.

Also the time taken by Question 1> Question 2 > Question 3

References:

- https://stackoverflow.com/questions/20577840/python-dictionary-sorting-in-descending-order-based-on-values
- https://stackoverflow.com/questions/22346086/measure-the-shell-script-execution-time-in-milliseconds-on-mac-os
- https://stackoverflow.com/questions/37697195/how-to-merge-two-data-frames-based-on-particular-column-in-pandas-python
- https://stackoverflow.com/questions/16923281/writing-a-pandas-dataframe-to-csv-file
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html
- https://stackoverflow.com/questions/6885441/setting-the-number-of-map-tasks-and-reduce-tasks
- https://codewitharjun.medium.com/
- https://blog.contactsunny.com/data-science/installing-hadoop-on-the-new-m1-pro-and-m1-max-macbook-pro
- https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/
-