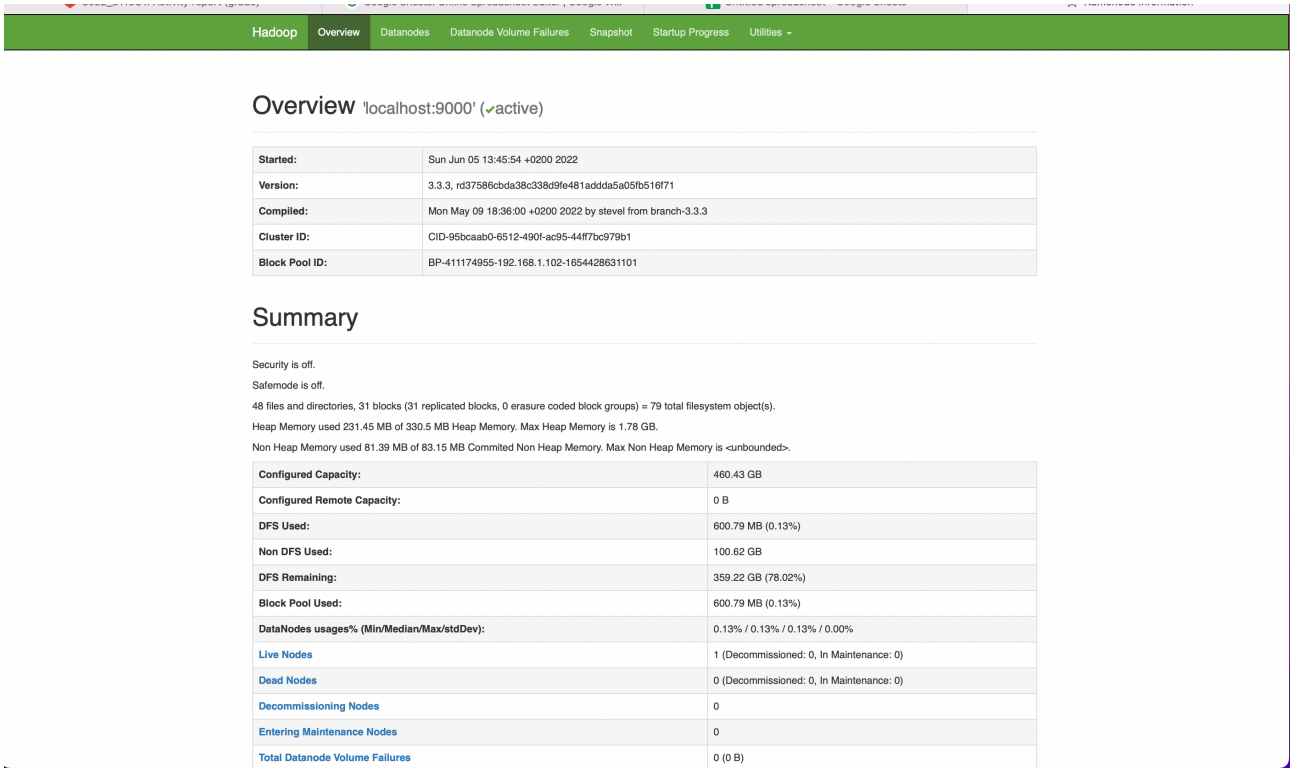
Sreyashi Saha Mat No: 1747271 Group1

Exercise 1: Preparing your Hadoop infrastructure

1. Hadoop User Interface in the localhost

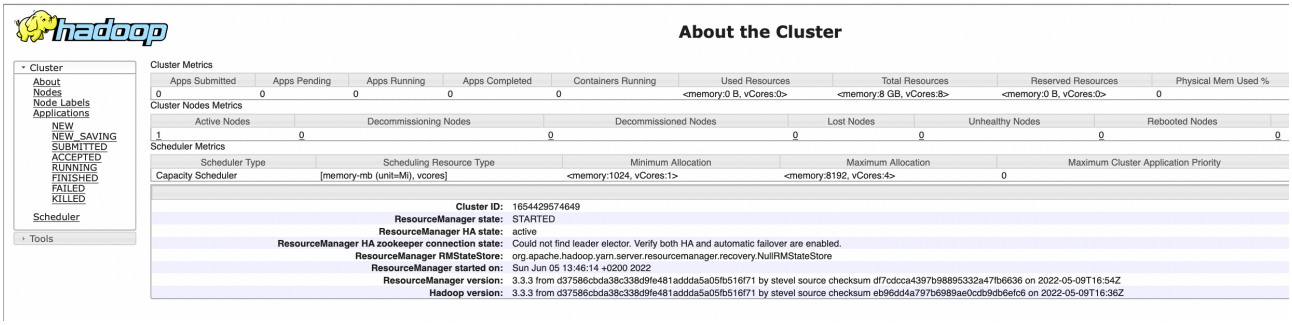


2. Check Hadoop version: hadoop version.

```
sreyashisaha@Sreyashis-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
sreyashisaha@Sreyashis-MacBook-Air ~ %
```

3. List files in HDFS: hadoop fs -ls /

```
sreyashisaha@Sreyashis-MacBook-Air ~ % hadoop fs -ls /
2022-06-09 18:12:53,068 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 4 items
drwxr-xr-x   - sreyashisaha supergroup          0 2022-06-08 14:18 /Question2
drwxr-xr-x   - sreyashisaha supergroup          0 2022-06-09 17:37 /Question3
-rw-r--r--   1 sreyashisaha supergroup   26827486 2022-06-05 17:19 /flight1.csv
-rw-r--r--   1 sreyashisaha supergroup     665057 2022-06-05 17:15 /text.txt
sreyashisaha@Sreyashis-MacBook-Air ~ %
```



The commands used to execute the word count program:

```
sreyashisaha@Sreyashis-MacBook-Air ~ % hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools
/lib/hadoop-streaming-3.3.3.jar -input /Question3/merged.csv -output /Question1/output -mapper mapper.py -redu
cer reducer.py -file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper.py -file /Users/sreyashi
saha/Desktop/Semester1/DDA/practical/Week6/reducer.py
```

In the mapper I read each line and extract each words from each line using strip and split commands. Then I print each word and assign the value as 1 for each word. That is my output from mapper is basically a dictionary with each word as the key and the each key is assigned the value 1. This output will further be used in my reducer as input to calculate the count of each word.

```
mapper.py > ...
1   #!/usr/bin/env python
2
3   import sys
4
5   for line in sys.stdin:
6       line = line.strip()
7       words = line.split()
8       for word in words:
9           print('%s\t%s' % (word, 1))
```

Of course we get a sorted and shuffled list as input for our reducer, which is done internally in HDFS.

In reducer I have taken several variables, current_word as none, count as 0 and word as none. Now when I read each line from the output of mapper I split the line and get the word and the count of the word from each input line. Now we covert the count value to string, since the output from mapper was a string value. Now comes the part where I check how many times a word is repeated and store that count. So basically, I check whether the current_word matches with the word that I get from input or not. If this condition satisfies then then we update the current_count with the count of that word. If the current word doesn't match with the input word then we print that word along with the count of the current word. Next to this step we update the current_word to the word that we obtained from the input an update the current_count to the initial count of the new word

reducer.py > ...

```python
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# read the entire line from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # slpiting the data on the basis of tab we have provided in mapper.py
    word, count = line.split('\t'),1
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

obtained. Now this process continues for all words till the second last word. For the last word we print it outside the for loop because there is no other word left to check the conditions with.

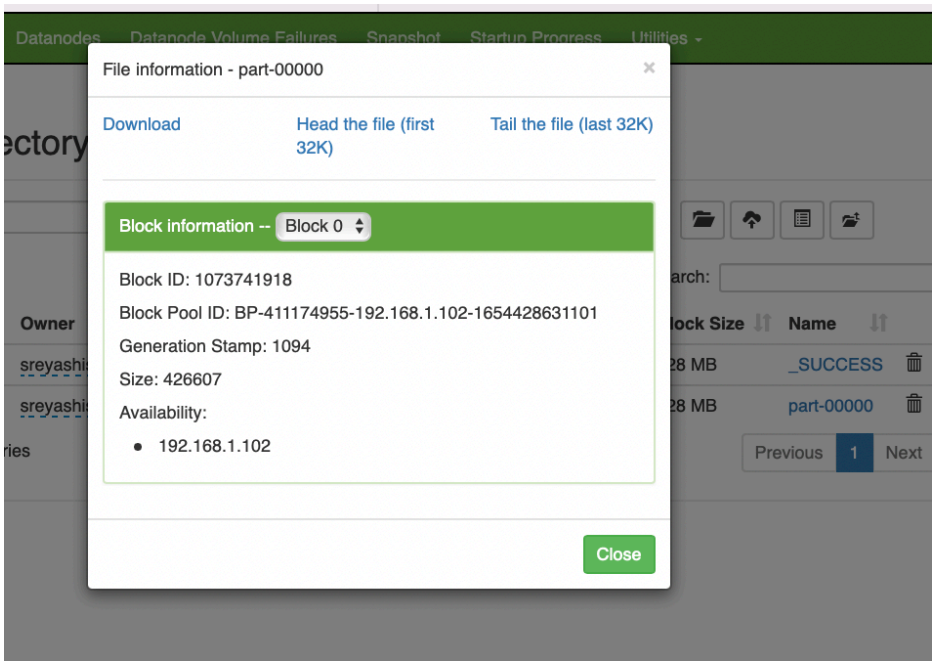
The command lines that I use to make a new directory and put the text.txt file for later computation are as follows:

```
sreyashisaha@Sreyashis-MacBook-Air ~ % hdfs dfs -mkdir /Question1
2022-06-09 18:39:44,308 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
sreyashisaha@Sreyashis-MacBook-Air ~ % hdfs dfs -put /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6
/text.txt /Question1/
2022-06-09 18:41:15,845 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
```

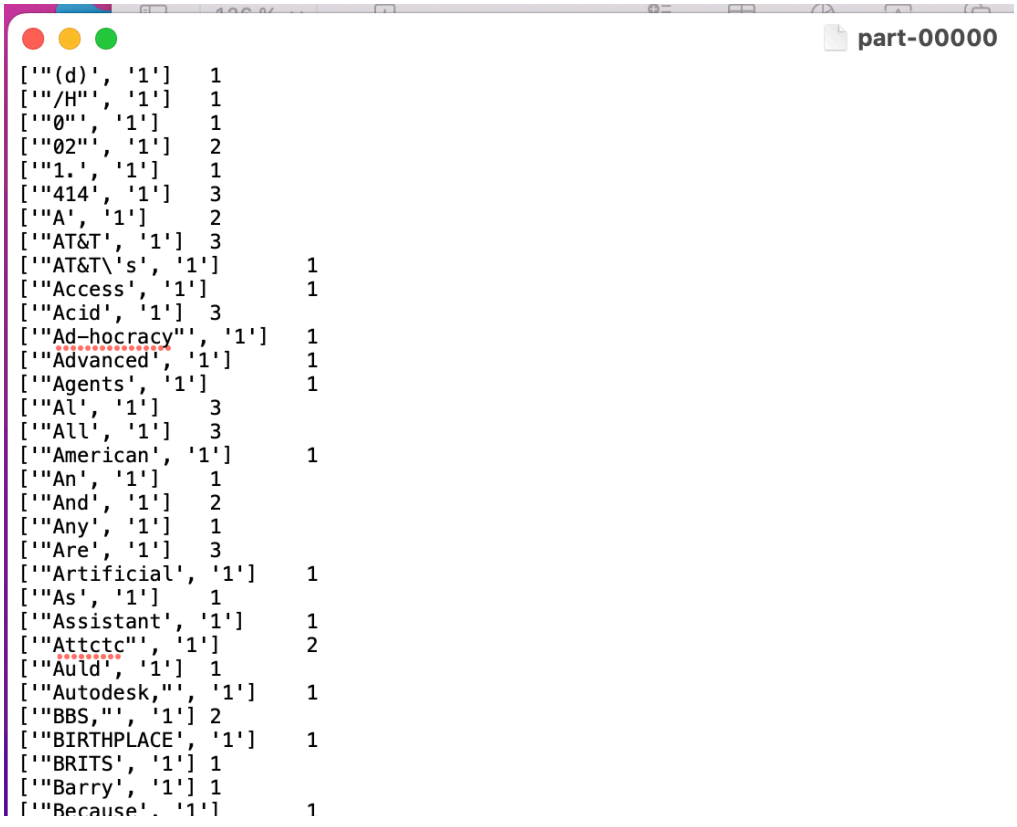
The command to execute in Hadoop Interface:

```
sreyashisaha@Sreyashis-MacBook-Air ~ % time hadoop jar /opt/
homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools/lib/
hadoop-streaming-3.3.3.jar
-input /Question1/text.txt
```

```
-output /Question1/output
-mapper mapper.py
-reducer reducer.py
-file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper.py
-file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/reducer.py
```



As we can see that an output has been stored in my Output folder.



This is a part of the output of the word count program.

The output of the console:

```
2022-06-09 18:41:15,845 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
sreyashisaha@Sreyashis-MacBook-Air ~ % time hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools/lib/hadoop-streaming-3.3.3.jar -input /Question1/text.txt -output /Question1/output -map
per mapper.py -reducer reducer.py -file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper.py -file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/reducer.py
2022-06-09 18:43:54,523 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
2022-06-09 18:43:54,771 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
packageJobJar: [/Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper.py, /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/reducer.py] □ /var/folders/mv/39f6vvvn1kb8pjc2zbvyxr5w0000gn
/T/streamjob4066661758539436673.jar tmpDir=null
2022-06-09 18:43:56,017 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2022-06-09 18:43:56,179 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2022-06-09 18:43:56,179 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2022-06-09 18:43:56,278 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2022-06-09 18:43:56,851 INFO mapred.FileInputFormat: Total input files to process : 1
2022-06-09 18:43:57,018 INFO mapreduce.JobSubmitter: number of splits:1
2022-06-09 18:43:57,281 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local85944858_0001
2022-06-09 18:43:57,281 INFO mapreduce.JobSubmitter: Executing with tokens: □
2022-06-09 18:43:57,611 INFO mapred.LocalDistributedCacheManager: Localized file:/Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper.py as file:/tmp/hadoop-sreyashisaha/mapred/local/job_local
85944858_0001_0f5d29a2-5f07-4a55-884b-2485d556a568/mapper.py
2022-06-09 18:43:57,694 INFO mapred.LocalDistributedCacheManager: Localized file:/Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/reducer.py as file:/tmp/hadoop-sreyashisaha/mapred/local/job_loca
l85944858_0001_015acb2f-812d-4d3d-97a8-29d5fe477ba4/reducer.py
2022-06-09 18:43:57,916 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2022-06-09 18:43:57,920 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2022-06-09 18:43:57,924 INFO mapreduce.Job: Running job: job_local85944858_0001
2022-06-09 18:43:57,929 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
2022-06-09 18:43:57,941 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-06-09 18:43:57,941 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-06-09 18:43:58,022 INFO mapred.LocalJobRunner: Waiting for map tasks
2022-06-09 18:43:58,026 INFO mapred.LocalJobRunner: Starting task: attempt_local85944858_0001_m_000000_0
2022-06-09 18:43:58,055 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-06-09 18:43:58,056 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-06-09 18:43:58,064 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on Linux.
2022-06-09 18:43:58,064 INFO mapred.Task:  Using ResourceCalculatorProcessTree : null
2022-06-09 18:43:58,079 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/Question1/text.txt:0+665057
2022-06-09 18:43:58,132 INFO mapred.MapTask: numReduceTasks: 1
2022-06-09 18:43:58,161 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2022-06-09 18:43:58,161 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2022-06-09 18:43:58,161 INFO mapred.MapTask: soft limit at 83886080
2022-06-09 18:43:58,161 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2022-06-09 18:43:58,161 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2022-06-09 18:43:58,164 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2022-06-09 18:43:58,181 INFO streaming.PipeMapRed: PipeMapRed exec [/Users/sreyashisaha/./mapper.py]
2022-06-09 18:43:58,187 INFO Configuration.deprecation: mapred.work.output.dir is deprecated. Instead, use mapreduce.task.output.dir
2022-06-09 18:43:58,191 INFO Configuration.deprecation: mapred.local.dir is deprecated. Instead, use mapreduce.cluster.local.dir
2022-06-09 18:43:58,191 INFO Configuration.deprecation: map.input.file is deprecated. Instead, use mapreduce.map.input.file
2022-06-09 18:43:58,192 INFO Configuration.deprecation: map.input.length is deprecated. Instead, use mapreduce.map.input.length
2022-06-09 18:43:58,192 INFO Configuration.deprecation: mapred.job.id is deprecated. Instead, use mapreduce.job.id
2022-06-09 18:43:58,192 INFO Configuration.deprecation: mapred.task.partition is deprecated. Instead, use mapreduce.task.partition
2022-06-09 18:43:58,193 INFO Configuration.deprecation: map.input.start is deprecated. Instead, use mapreduce.map.input.start
2022-06-09 18:43:58,193 INFO Configuration.deprecation: mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
2022-06-09 18:43:58,193 INFO Configuration.deprecation: mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.id
2022-06-09 18:43:58,193 INFO Configuration.deprecation: mapred.tip.id is deprecated. Instead, use mapreduce.task.id
2022-06-09 18:43:58,193 INFO Configuration.deprecation: mapred.skip.on is deprecated. Instead, use mapreduce.job.skiprecords
2022-06-09 18:43:58,194 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.name
2022-06-09 18:43:58,318 INFO streaming.PipeMapRed: R/W/S=1/0/0 in:NA [rec/s] out:NA [rec/s]
2022-06-09 18:43:58,318 INFO streaming.PipeMapRed: R/W/S=10/0/0 in:NA [rec/s] out:NA [rec/s]
2022-06-09 18:43:58,319 INFO streaming.PipeMapRed: R/W/S=100/0/0 in:NA [rec/s] out:NA [rec/s]
```

```
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=426607
2022-06-09 18:44:00,218 INFO mapred.LocalJobRunner: Finishing task: attempt_local85944858_0001_r_000000_0
2022-06-09 18:44:00,218 INFO mapred.LocalJobRunner: reduce task executor complete.
2022-06-09 18:44:00,979 INFO mapreduce.Job:  map 100% reduce 100%
2022-06-09 18:44:00,984 INFO mapreduce.Job: Job job_local85944858_0001 completed successfully
2022-06-09 18:44:01,020 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=2162408
                FILE: Number of bytes written=4542872
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1330114
                HDFS: Number of bytes written=426607
                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=13006
                Map output records=105488
                Map output bytes=867930
                Map output materialized bytes=1078912
                Input split bytes=92
                Combine input records=0
                Combine output records=0
                Reduce input groups=21438
                Reduce shuffle bytes=1078912
                Reduce input records=105488
                Reduce output records=21438
                Spilled Records=210976
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=630194176
        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=665057
        File Output Format Counters
                Bytes Written=426607
2022-06-09 18:44:01,022 INFO streaming.StreamJob: Output directory: /Question1/output
hadoop jar  -input /Question1/text.txt -output /Question1/output -mapper       10.66s user 1.52s system 157% cpu 7.748 total
sreyashisaha@Sreyashis-MacBook-Air ~ %
```

Exercise 2: Analysis of Airport efficiency with Map Reduce (10 points)

In this exercise I downloaded data from Bureau of Transportation Statistics' homepage. On the Bureau' homepage I downloaded data by selecting following fields:

• Filter geography: all • Filter year: 2017
• Filter period: January
• Time period: FlightDate
• Airline: Reporting Airline, Flight Number Reporting Airline • Origin: Origin
• Destination: Dest
• Departure Performance: DepTime, DepDelay
• Arrival Performance: ArrTime, ArrDelay

After the selection I got a CSV file containing 450017 lines. An example of a line is as follows

1/1/2017 12:00:00 AM,AA,307,DEN,PHX,1135,-10,1328,-17

Now before execution of the program let us understand what Map_reduce actually is in Hadoop.

MAP_REDUCE:

Map-Reduce is a programming model with two phases: the Map Phase and the Reduce Phase. It is intended for parallel processing of data that is distributed across multiple machines (nodes). Mapper and Reducer are the two Hadoop Python scripts. MapReduce divides a task into subtasks, handles the subtasks in parallel, and then combines the outputs of the subtasks to generate the final output.

A mapper is a function that processes the data it receives. The mapper divides the data into small chunks and analyses it. Even if the input to a MapReduce program is a file or directory, the input to the mapper function is in the form of (key, value) pairs (which is stored in the HDFS).

Working Of The Mapper:

1) The Mapper receives the input data from the users, which is formatted using an InputFormat. It specifies the location of the input data on HDFS as a file or directory. It also determines how the input data should be split into input splits.
2) Each Mapper is responsible for a single input split. RecordReader are InputFormat objects that are used to extract (key, value) records from an input source (split data).
3) The Mapper takes the (key, value) pairs as input and produces (key, value) pairs as output. The intermediate output is the result of the Mapper.
4) The input key may be used or ignored entirely by the Mapper. A standard pattern, for example, is to read a file one line at a time. The byte offset into the file where the line begins is the key. The line's content serves as the value. Normally, the key is regarded as unimportant. The output of the Mapper must be in the form of key/value pairs if it writes anything.
5) The Reducer receives the Mapper's output (intermediate keys and their value lists) in sorted key order.
6) The Reducer generates a number of final key/value pairs, which can be zero or more. These are stored in HDFS. For each input key, the Reducer normally emits a single key/value pair.
7) If a Mapper appears to be running slower or laggy than the others, a new instance of the Mapper will be launched on a different machine and run on the same data. The outcomes of the first Mapper to complete the task will be used. Hadoop will replace the Mapper, which is currently in use.

The number of map tasks in a MapReduce application is determined by the size of the input file's data blocks. For example, if each block of divided data is 128MB in size and the input data is 1GB in size, the number of map tasks will be eight. As the amount of incoming data grows, so does the number of map tasks, resulting in increased parallelism and faster data processing.

REDUCER:

A Reducer is a function that takes the Mapper's output (intermediate key-value pair) and processes each one separately to generate the output. This data (key, value) can be aggregated, filtered, and combined in a variety of ways for a variety of processes. The reducer's output is the final output, which is saved in HDFS.

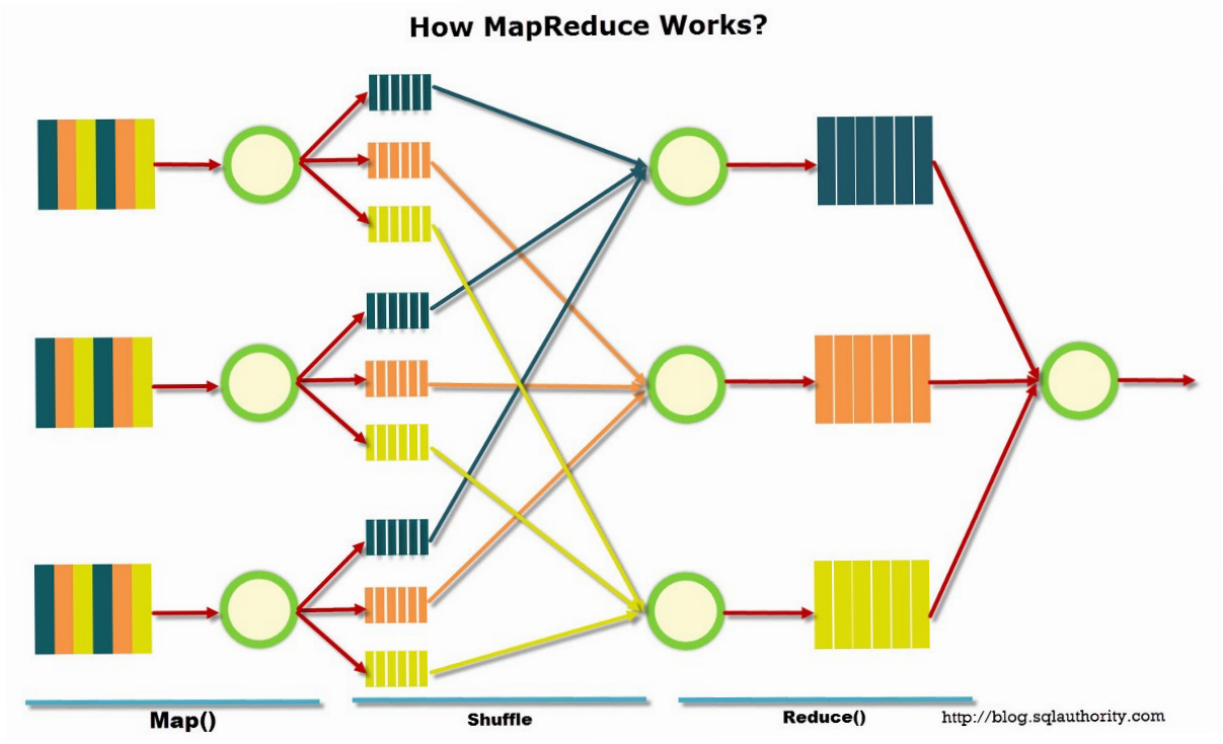
SHUFFLE AND SORT:

In Hadoop, the shuffle phase passes map output from a Mapper to a Reducer in MapReduce. In MapReduce, the sort phase is responsible for merging and sorting map outputs. The mapper's data is aggregated by key, distributed among reducers, then sorted by key. All values associated with the same key are obtained by each reducer. Hadoop's shuffle and sort phases happen at the same time and are handled by the MapReduce framework.

Shuffling: Shuffling is the process of transferring data from mappers to reducers, or the process by which the system sorts the map output and passes it on to the reducer as input. As a result, the reducers require the MapReduce shuffle step; otherwise, they would have no input (or input from every mapper). Because shuffling can begin before the map phase is completed, it saves time and allows the chores to be completed faster.

Sorting: The keys supplied by the mapper are automatically sorted by the MapReduce Framework, i.e., all intermediate key-value pairs in MapReduce generated by the mapper are sorted by key and not by value before commencing the reduction. Each reducer's values are not ordered; they can be in any sequence.

Sorting in Hadoop makes it easier for reducers to determine when a new reduce process should begin. The reducer will save time as a result of this. When the next key in the sorted input data is different from the preceding, the reducer starts a new reduce process. Each reduce task receives key-value pairs as input and outputs key-value pairs.



SOLUTION 2:

1. Computing the maximum, minimum, and average departure delay for each airport.[Hint: you are required to find max, min and avg in a single map reduce job]

   When I run my mapper and reducer locally I get the following output:

```
sreyashisaha@Sreyashis-MacBook-Air Week6 % python3 mapper1.py
< flight1.csv | sort | python3 reducer1.py
Origin    Maximum    Minimum    Average

ABE        794       -11       19.822580645161292
ABI        263       -11       28.153846153846153
ABQ        911       -18       8.67610748002905
ABR        1259       -13       38.101694915254235
ABY        291       -21       10.275
ACT        202       -17       13.896907216494846
ACV        578       -17       11.142857142857142
ACY        670       -21       8.193146417445483
ADK        41       -34       2.375
ADQ        73       -28       -6.256410256410256
AEX        354       -20       11.024590163934427
AGS        1130       -11       21.881720430107528
ALB        981       -20       11.99468791500664
AMA        298       -15       4.528925619834711
ANC        1195       -48       3.047818791946309
APN        278       -19       22.414634146341463
ASE        1110       -20       35.817275747508305
ATL        1470       -22       15.202814794669568
ATW        1065       -16       26.742424242424242
AUS        1246       -22       8.71161657189277
AVL        425       -14       15.356435643564357
AVP        783       -18       33.191489361702125
AZO        268       -20       10.347826086956522
BDL        549       -20       6.674255691768827
BET        72       -27       -2.6973684210526314
BFL        290       -20       -2.5689655172413794
```

Now I run the same files in my HDFS and get the following results:

```
hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/tools/
lib/hadoop-streaming-3.3.3.jar
-input /flight1.csv
-output /Question2/output
-mapper mapper1.py
```

-reducer reducer1.py
-file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/mapper1.py
-file /Users/sreyashisaha/Desktop/Semester1/DDA/practical/Week6/reducer1.py

This is the command that I used to execute my mapper and reducer in Hadoop.

We can see that we have the same output for running the codes in local system as well as in HDFS.

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

```
sreyashisaha@Sreyashis-MacBook-Air Week6 % python3 mapper1_2.p
y < flight1.csv | sort | python3 reducer2_2.py
Airport          Avg Arrival Delay
LAW              89.03846153846153

GGG              76.8

EKO              64.14583333333333

BPT              61.333333333333336

LWS              43.71739130434783

ABR              35.644067796610166

ASE              32.91029900332226

HDN              31.937142857142856

ABI              30.5

ELM              30.153846153846153
```

These are the top 10 airports by their Average Arrival Delay when run on local system.

The output that I get when I run it on hdfs is as follows:

```
hadoop jar /opt/homebrew/Cellar/hadoop/3.3.3/libexec/share/hadoop/
tools/lib/hadoop-streaming-3.3.3.jar
-input /flight1.csv
-output /Question2/output1
-mapper mapper1_2.py
```

```
-reducer
reducer2_2.py
-file /Users/
sreyashisaha/
Desktop/Semester1/
DDA/practical/Week6/
mapper1_2.py
-file /Users/
sreyashisaha/
Desktop/Semester1/
DDA/practical/Week6/
reducer2_2.py
```

3. What are your mapper.py and reduce.py solutions?
4. Describe step-by-step how you apply them and the outputs during this procedure.

The mapper and reducer solutions along with the explanations are as follows:

mapper1.py > ...

```python
#!/usr/bin/env python

import sys
import csv

file = csv.reader(sys.stdin,delimiter=',')
# print(file)
next(file)
for line in file:
    FL_DATE, OP_UNIQUE_CARRIER, OP_CARRIER_FL_NUM, ORIGIN, DEST, DEP_TIME, DEP_DELAY, ARR_TIME, ARR_DELAY = line
    print('%s\t%s' % (ORIGIN, DEP_DELAY))

```

The above code is for Mapper1_1 (question 2 part 1). In this I read my dataset using csv reader and extract each and every row of the dataset. Now, since the dataset has column names included as it's very first row therefore, I have used the command next(file) to skip the 1st row and read the data from the second row. Now my row has total 9 values, thus I have assigned those 9 values to the respective column names. From these 9 values my mapper just prints the airport origin and the departure delay time, which will be further used in reducer for computation. So the output of a mapper is in key/value format. In my case the key is the origin and the value is the Departure Delay.

After the mapping phase the output file from mapper is sorted and presented to the reducer.

reducer1.py > ...

~/Desktop/Semester1/DDA/practical/Week6/reducer1.py

```python
import sys
import csv
delay_list = []
current_airport = None
print("Origin",'  ',"Maximum",'  ',"Minimum",'  ',"Average",'\n')
for line in sys.stdin:
    # print("line is",line)
    line = line.strip()
    origin, delay = line.split('\t')
    if current_airport is not None:
        if current_airport == origin:
            delay_list.append(int(float(delay)))
        else:
            if delay_list:
                print(current_airport,'      ', max(delay_list),'    ', min(delay_list),'    ',(sum(delay_list)/len(delay_list)))
            current_airport=origin
            delay_list=[]
    else:
        current_airport=origin
        delay_list.append(int(float(delay)))

```

In the above code for Reducer1_1(question 2 part1) I get my input from my mapper file that I run before. So basically my input data is the origin and the departure delay. I read this data as each line and then I split the data obtained in each line so that now I have two separate values to work with. Now I have to find the maximum, minimum and average of departure delay for each of the airport origins. So I take a variable named Current Airport and assign it as NONE. Now in the first checking the condition that I provide doesn't satisfy, therefore, it goes to the else part and updates the current airport to the one that we got as input for the first time and appends the departure delay value for that origin to the delay list. In the next iteration my first if condition satisfies thus I go ahead and I check if this current airport matches the origin that I receive as input. If it finds a match then I

append the departure delay values for the matched origin to a list. It will keep on appending until the origin received from the mapper changes. Now when the current airport doesn't match with the origin then I check if the delay list is not empty, and we do not get an empty list because we have appended all the values for a particular origin. Next to this step I calculate the maximum, minimum and average departure delay for the current airport and print it. I then update my current airport to the origin I received as the input and empty the delay list so that new values can be appended.

This process is continued until we get the minimum, maximum and average for all the airports.

mapper1_2.py > ...

```python
#!/usr/bin/env python

import sys
import csv

file = csv.reader(sys.stdin,delimiter=',')
# print(file)
next(file)
for line in file:
    FL_DATE, OP_UNIQUE_CARRIER, OP_CARRIER_FL_NUM, ORIGIN, DEST, DEP_TIME, DEP_DELAY, ARR_TIME, ARR_DELAY = line
    print('%s\t%s' % (ORIGIN, ARR_DELAY))
```

The above code is for Mapper1_1 (question 2 part 1). In this I read my dataset using csv reader and extract each and every row of the dataset just like the last one. In here my mapper gives my reducer the airport names and the arrival delay instead of departure delay, i.e. here my key is origin and its corresponding value is the arrival delay.

Now for the reducer, I use a similar process as before. Here, my input data is the origin and the arrival delay. I read this data as each line and then I split the data obtained in each line so that now I have two separate values to work with. Now I have to find the average of arrival delay for each of the airport origins and print the top 10. So I take a variable named Current Airport and assign it as NONE. Now in the first checking the condition that I provide doesn't satisfy, therefore, it goes to the else part and updates the current airport to the one that we got as input for the first time and appends the arrival delay value for that origin to the delay list. In the next iteration my first if condition satisfies thus I go ahead and I check if this current airport matches the origin that I receive as input. If it finds a match then it will keep on appending until the origin received from the mapper changes. Now when the current airport doesn't match with the origin then I check if the delay list is not empty, and we do not get an empty list because we have appended all the values for a particular origin. Next to this step I append the average of arrival delay values for the matched origin to a list named average and I also append the current airport value to a list airport. I then update my current airport to the origin I received as the input and empty the delay list so that new values can be appended. When I have all the averages for all the airports, then I zip the airport list and average list together. The zip() function

returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc. Then I sort the zipped list according to the average arrival delay. The sorted() function sorts the elements of a given zipped list in a specific order (ascending or

reducer2_2.py > ...

```python
#!/usr/bin/env python
import sys
import csv
delay_list = []
current_airport = None
airport=[]
average =[]

for line in sys.stdin:
    # print("line is",line)
    line = line.strip()
    origin, delay = line.split('\t')
    if current_airport is not None:
        if current_airport == origin:
            delay_list.append(int(float(delay)))
        else:
            if delay_list:
                airport.append(current_airport)
                average.append((sum(delay_list)/len(delay_list)))
                # print(current_airport, max(delay_list), min(delay_list),(sum(delay_list)/len(delay_lis
                current_airport=origin
                delay_list=[]
    else:
        current_airport=origin
        delay_list.append(int(float(delay)))

sort_function = zip(airport, average)
sorted_values = sorted(sort_function, key=lambda val: val[1], reverse=True)[:10]
# print(sorted_values)
print("Airport",'\t',"Avg Arrival Delay",'\n')
for key in sorted_values:
    print(*key, sep='\t\t')
    print('\n')

```

descending) and returns it as a list. Since I want the top 10 whose average arrival delay is maximum therefore, my key argument in the sorted function sorts values according to the second place in the list. Th reverse = True argument sorts the list in descending order and returns the top 10.

Exercise 3: Analysis of Movie dataset using Map and Reduce (10 points)

For this exercise you will use movielens10m dataset available at http://files.grouplens.org/ datasets/ movielens/ml-10m-README.html. The movielens10m dataset consists of 10 million rating entries by users. There are two main files required to solve this exercise 1) rating.dat and 2) movie.dat. The rating.dat file contains userId, movieId and ratings (on scale of 1 to 5). The movie.dat file contains information about movies i.e. movieId, Title and Genres.

In this exercise, you are going to write a python code using Hadoop MapReduce to accomplish several requirements, note as your dataset is large you also need to perform some performance analysis i.e. how many mappers and reducers you used, what is the performance increase by varying number of mappers and reducers. Please also note that although you may have very few physical cores i.e. 2 or 4 or 8, but you can still experiment with a large number of mappers and reducers. Plot the execution time vs the number of mappers × reducers for each task below.

SOLUTION:

First before going to the map reduce phase, we will have to merge the two files in a preprocessing step.

```
import pandas as pd
movies_columns = ["movieId", "title", "genre"]
ratings_columns = ["userId", "movieId", "rating", "timestamp"]
df_movies = pd.read_csv('movies.dat', sep='::', names=movies_columns)
print(df_movies.shape)
df_ratings = pd.read_csv('ratings.dat', sep='::', names=ratings_columns)
pd.set_option('display.max_columns', None)
#

# df = pd.concat([df_ratings, df_movies], join="outer", axis=1).drop(['timestamp', 'movieId'], axis=1)
df_merged = df_ratings.merge(df_movies, on="movieId").drop(['timestamp'], axis=1)
# print(df)
df_merged.sort_values('movieId')
df_merged.to_csv('merged.csv', index=False)
```

After this step I get a merged dataset that I stored in merged.csv file for use in map_reduce phase. Note that I have sorted my data frame according to the movie id and then converted it into a csv file. My merged csv somewhat looks like this:

1. Find the movie title which has the maximum average rating?
   For this question my mapper extracts the movie title and the ratings which is the

mapper3_1.py > ...

```
#!/usr/bin/env python

import sys
import csv
file = csv.reader(sys.stdin,delimiter=',')
next(file)

for line in file:
    print('%s\t%s'%(line[0], line[3]))
```

```
Jumanji (1995),Adventure|Children|Fantasy,2,3.0,65
```

0th and 3rd position in each row. This is provided to my reducer for further computation.

In HDFS again sorting is done internally and therefore, we receive a sorted file as input for the reducer.

In the reducer below I initialise a dictionary named rating_dict. Now I read the output of the mapper as my reducer's input. For this question I get movie name and the ratings as my input which I split. Now I check whether my movie that I get as input is already

reducer3_1.py > ...

```
#!/usr/bin/env python

import sys

rating_dict = {}
for line in sys.stdin:
    # print(line)
    movie, ratings = line.split('\t')
    # print(movie,ratings)
    ratings = ratings.strip()
    if movie in rating_dict:
        rating_dict[movie].append(int(float(ratings)))
    else:
        rating_dict[movie]=[]
        rating_dict[movie].append(int(float(ratings)))

for key,value in rating_dict.items():
    rating_dict[key]=sum(value)/len(value)

sorted_values = sorted(rating_dict.items(), key=lambda x: x[1], reverse=True)[:5]

print("Movie Id",'\t\t\t\t',"Avg Ratings",'\n')
for key in sorted_values:
    print(*key, sep='\t\t')
    print('\n')
```

present as a key in my dictionary rating_dict. For the first iteration it is not present, therefore, it goes to the else part. In the else part it creates an empty list for the key movie(input), and also appends the rating value for that particular movie. In the next iteration say the movie which is read from the input exists in the dictionary, then it

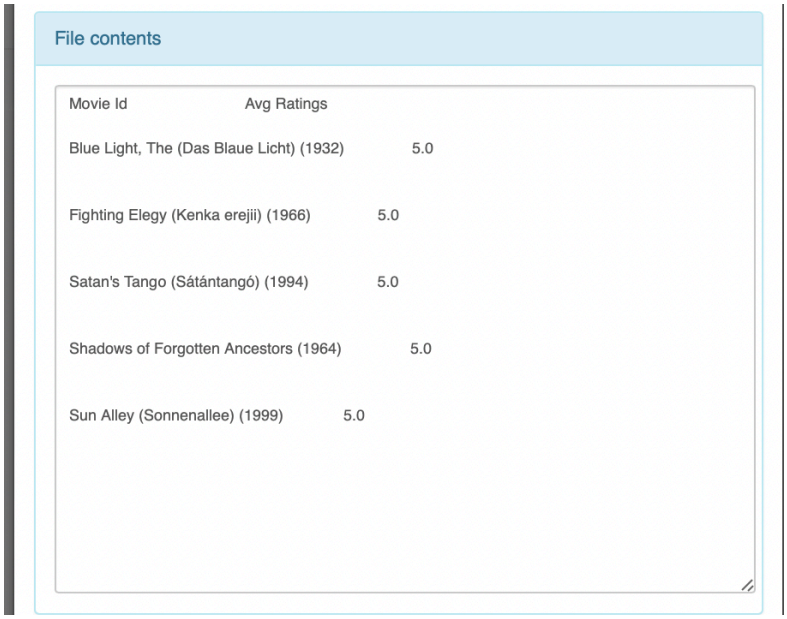
appends the ratings corresponding to that movie. Like this for each movie title we obtain a list of ratings corresponding to each movie. After this step I start a for loop and find the average of ratings for each key in the dictionary, i.e. for each movie.

After finding the average I sort the dictionary using the sorted command and print the top five movies in descending order of the averages.

When I run it locally I get the following output:

```
sreyashisaha@Sreyashis-MacBook-Air Week6 % python3 mapper3_1.py < merged.csv | sor
t | python3 reducer3_1.py
Movie Id                                Avg Ratings

Blue Light, The (Das Blaue Licht) (1932)                5.0


Fighting Elegy (Kenka erejii) (1966)            5.0


Satan's Tango (Sátántangó) (1994)               5.0


Shadows of Forgotten Ancestors (1964)           5.0


Sun Alley (Sonnenallee) (1999)          5.0


sreyashisaha@Sreyashis-MacBook-Air Week6 %
```

On running the mapper and reducer in HDFS, I get the following output:



So from this we can see that highest average movie is Blue Light, The (Das Blue Licht) (1932) with a max rating of 5.

2. Find the user who has assign lowest average rating among all the users who rated more than 40 times?
   For this question my mapper extracts the userID and the ratings which is the 4th and 3rd position in each row. This is provided to my reducer for further computation.

```python
#!/usr/bin/env python
import sys
import csv

reader = csv.reader(sys.stdin, delimiter=',')
next(reader)
for line in reader:
    print(line[4], '\t', line[3])
```

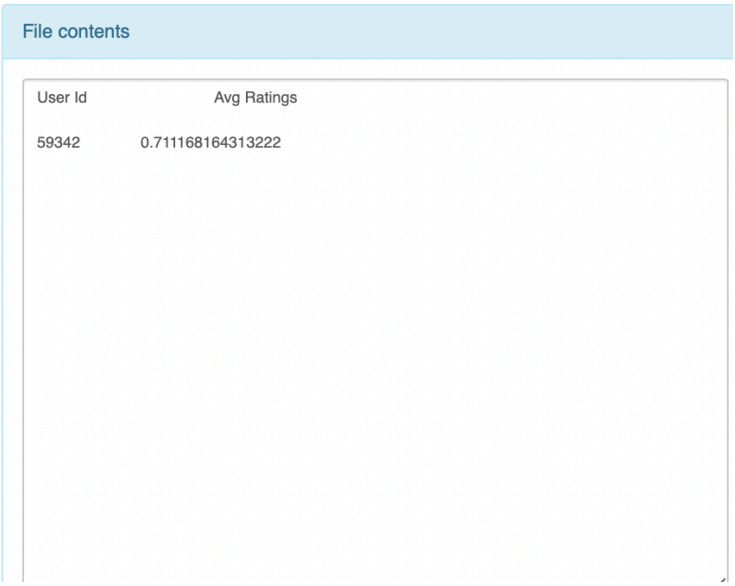
In the reducer below I have used the same logic to append the ratings of each user id in a dictionary, i.e. my dictionary has key userid and the list of ratings as our value for that particular id. After this I run a for loop and check which user has rated more than 40 times and find the average rating for that user and again append the userid and average found to two separate lists. Next I zip the list and sort them and finally print the User Id with maximum average rating.

```python
#!/usr/bin/env python
import sys

user_rating_dict={}
# user_ratings_list = []
# current_user = None
user_list = []
average = []


for line in sys.stdin:
    userid, ratings = line.split('\t')
    ratings = ratings.strip()
    if userid in user_rating_dict:
        user_rating_dict[userid].append(int(float(ratings)))
    else:
        user_rating_dict[userid]=[]
        user_rating_dict[userid].append(int(float(ratings)))
# print(user_rating_dict)

for user in user_rating_dict:
    length_userid = len(user_rating_dict[user])
    if length_userid>40:
        user_list.append(user)
        avg = sum(user_rating_dict[user])/len(user_rating_dict[user])
        average.append(avg)

sort_function = zip(user_list, average)
sorted_values = sorted(sort_function, key=lambda val: val[1], reverse=False)[:1]

print("User Id",'\t\t',"Avg Ratings",'\n')
for key in sorted_values:
    print(*key, sep='\t\t')
    print('\n')
```

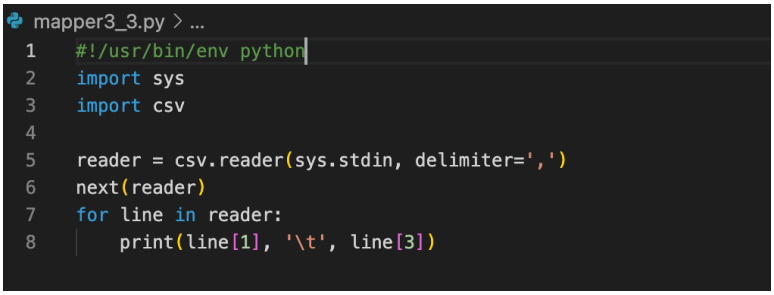
When I run the map reduce code locally I get the following output:

```
sreyashisaha@Sreyashis-MacBook-Air Week6 % python3 map
per3_2.py < merged.csv | sort | python3 reducer3_2.py
User Id                 Avg Ratings

59342           0.711168164313222


sreyashisaha@Sreyashis-MacBook-Air Week6 %
```

This is the output that I receive when I run it on HDFS.



3. Find the highest average rated movie genre? [Hint: you may need to merge/ combine movie.dat and rating.dat in a preprocessing step.]
   For this question my mapper extracts the genre of the movies and the ratings which is the 1st and 3rd position in each row. This is provided to my reducer for further computation.



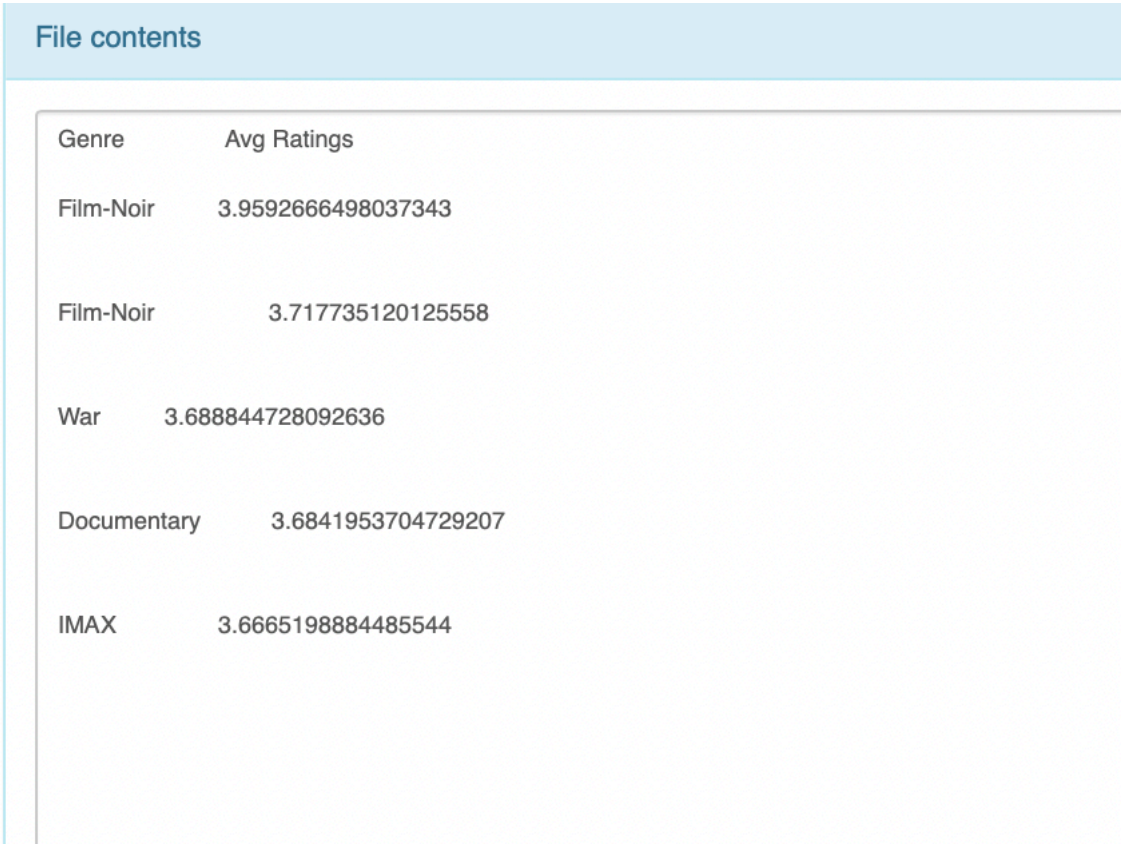
In the reducer below after I have the genre and movie ratings as input from mapper, I split the genre wrt 'l' as seen in my merged file. Now for I create a dictionary which stores the a list of ratings for each genre. Then I run a for loop on the dictionary that I created to find the average ratings of each genre and append the genre and average of that genre to two separate lists. Next to this step I sort the zipped list created and print the average ratings for the top 5 genres.

```python
#!/usr/bin/env python
import sys

genre_rating_dict={}
genre_average_rating_dict={}
# user_ratings_list = []
# current_user = None
genre_list = []
average = []


for line in sys.stdin:
    genre, ratings = line.split('\t')
    ratings = ratings.strip()
    # print(genre, ratings)
    genre_parts = genre.split('|')
    for part in genre_parts:
        genre = part
        if genre in genre_rating_dict:
            genre_rating_dict[genre].append(int(float(ratings)))
        else:
            genre_rating_dict[genre]=[]
            genre_rating_dict[genre].append(int(float(ratings)))

for genre_type in genre_rating_dict:
    genre_list.append(genre_type)
    avg = sum(genre_rating_dict[genre_type])/len(genre_rating_dict[genre_type])
    average.append(avg)

sort_function = zip(genre_list, average)
sorted_values = sorted(sort_function, key=lambda val: val[1], reverse=True)[:5]
print("Genre",'\t\t',"Avg Ratings",'\n')
for key in sorted_values:
    print(*key, sep='\t\t')
    print('\n')
```

When I run my code locally this is the output that I get:

```
sreyashisaha@Sreyashis-MacBook-Air Week6 % python3 mapper3_3.py < merged.csv | sort | python3 reduce
r3_3.py
Genre            Avg Ratings

Film-Noir               3.9592666498037343


Film-Noir               3.717735120125558


War             3.688844728092636


Documentary             3.6841953704729207


IMAX            3.6665198884485544


sreyashisaha@Sreyashis-MacBook-Air Week6 %
```

After I run the same on HDFS I get the following results:



Plot the execution time vs the number of mappers × reducers for each task above.

Mapper	Reducer	Time
1	1	56.75
2	2	56.43
4	4	57.87
6	6	58.63
8	8	59.82

Table For Exercise 3_1

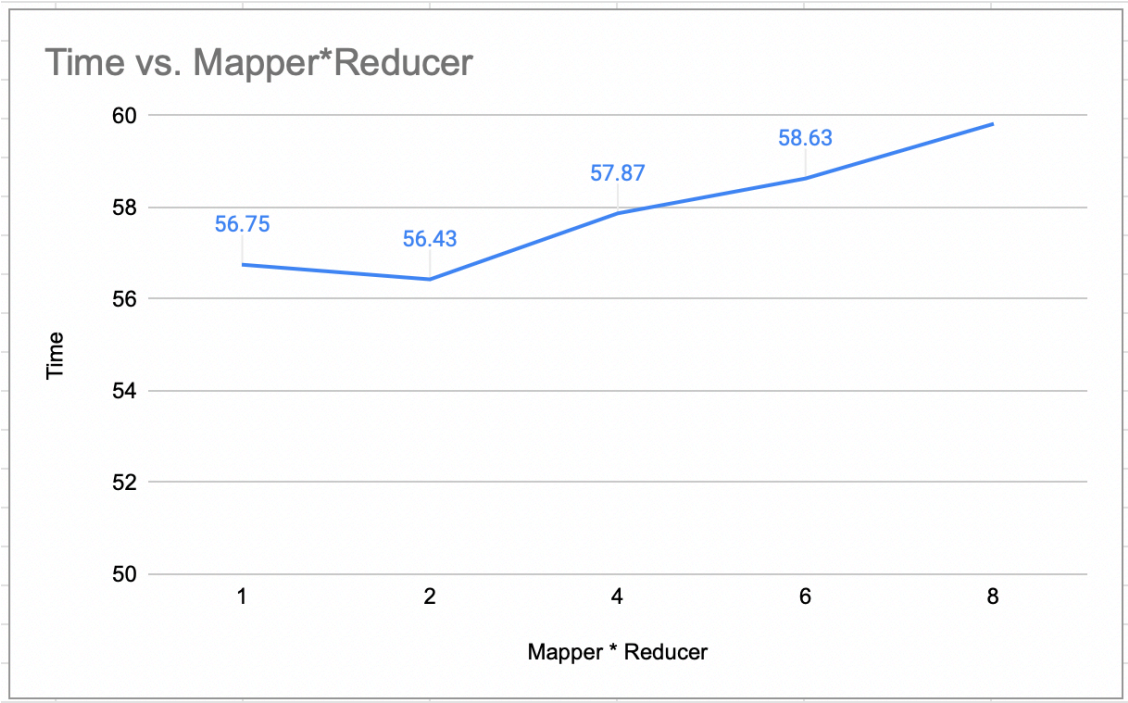
Table For Exercise 3_2

Mapper	Reducer	Time
1	1	60.57
2	2	61.09
4	4	63.07
6	6	63.82
8	8	67.02

Mapper	Reducer	Time
1	1	75.97
2	2	77.09
4	4	77.61
6	6	78.13
8	8	80.17

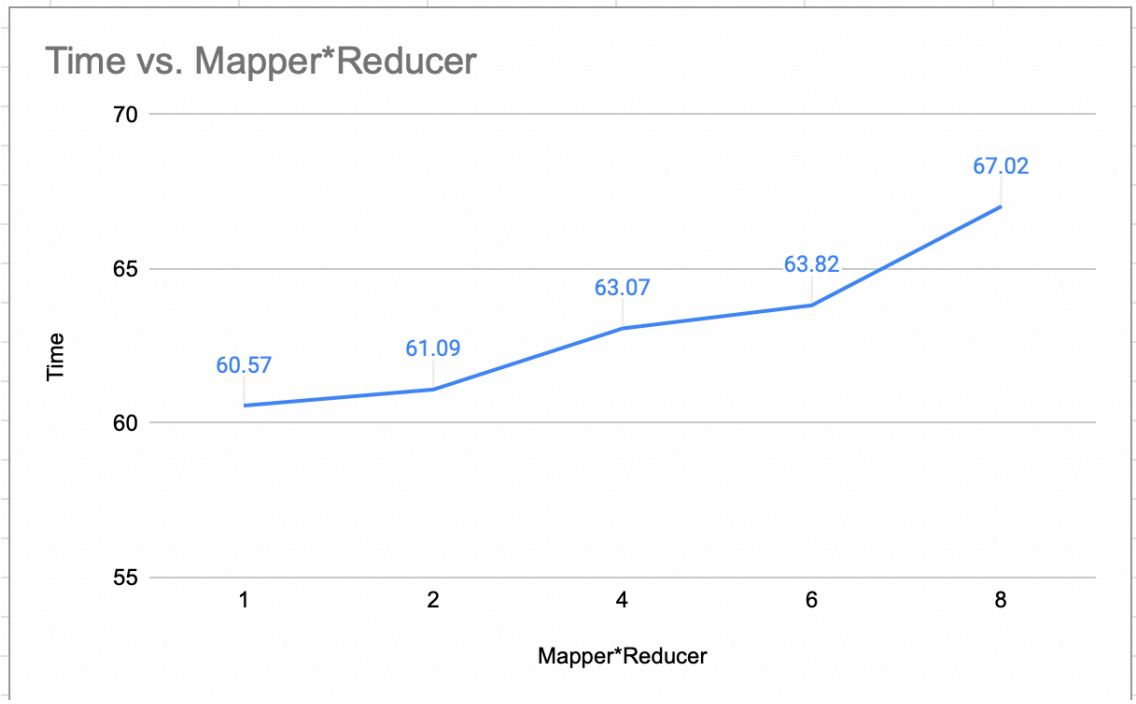
Table For Exercise 3_3

Graph for Exercise 3_1



We can see from this graph that overall the time is increasing as I increase my number of mappers and reducers.

Graph for Exercise 3_2



Here also we see an increase in the time as I increase the number of reducers and mappers.

Graph for Exercise 3_3

Here again we can see from this graph that overall the time is increasing as I increase my number of mappers and reducers.