COSC 6339
Big Data Analytics
Homework – 1

Priscilla Imandi
1619570

Problem Description

To find the percentage of delayed flights from a file of 21 Million flights data

i) Based on the Origin airport of the flight

ii) Based on the Origin airport and the month of the flight

Solution Explanation

i) <u>Based on Origin Airport of the flight</u>

Mapper :

```
class Mapper(api.Mapper):
    def map(self, context):
        data = context.value.split(",")
        # data[14] => delay time
        # data[16] => origin airport code
        key_value = data[16]
        # If there is no data we assume that the flight is on time
        if((data[15] == 'NA') or (data[15] == '') or (data[15] == ' ')) :
            temp = 0.0
        elif(re.search('[a-zA-Z]', data[15])):
            temp = -1.0
        else:
            #print(data[15])
            temp = float(data[15])

        if(temp > 0):
            context.emit(key_value, 1)
        else:
            context.emit(key_value, 0)
```

- Every mapper function will receive a single flight record as the input.
- Since, it is a csv file we will split the record based on the delimiter ‘,’.
- Now when we take a look at the data file, we can see that the 15th field is the delay time and the 17th field is the origin airport.
- Now if the delay is greater than 0 that means that the flight has arrived late.
- Whenever there is a delay we omit the value 1 from the mapper along with the key.
- The key here is the Origin airport code.

Reducer:

```
class Reducer(api.Reducer):
    def reduce(self, context):
        #s = sum(context.values)
        numDelays = 0
        totalFlights = 0
        for value in context.values:
            numDelays = numDelays + value
            totalFlights = totalFlights + 1
        percent_delayed_flights = ((float(numDelays) * 100) /float( totalFlights))
        context.emit(context.key, percent_delayed_flights)
```

- The reducer will receive the key value pair – Origin airport code, 0/1 based on whether a delay has occurred or not.
  Example : (IAH, 1) – Origin from IAH and a delay has occurred.
- Now the reducer will sum up all the values of 1 – meaning it counts the number of delays.

- Also, a counter to count the total number of values per key is used – this lets us the total number of flights for that origin airport.

- After calculating the percentage of flights the reducer outputs the value in the output file.
- All the values from one key will be directed to one reducer.

Output File:

```
bigd29@whale:/home2/cosc6339/bigd29/source> hdfs
dfs -cat /bigd29/output/output4/part-r-00000
ACY     31.0643564356
ALB     36.2681201438
APF     34.8630643967
AUS     36.3546561482
AVL     33.608815427
BHM     35.870433145
BLI     27.2157564906
BQN     26.188752112
BRO     23.1986594675
BRW     53.6985018727
BUR     40.3808300354
CAE     35.9287688056
CMI     25.6291090456
COS     30.6299524564
CSG     39.3925011865
DAL     52.4027997233
```

- A part of the output file can be seen above.
- The percentage is taken as float.
- The location of the output file is **/bigd29/output/output4/part-r-00000** on the HDFS file system.
- If only one reducer is considered – all the computations are done by 1 reducer.
- If multiple reducers are considered then they form subdirectories **part-r-00001** and so on.

ii) **<u>Based on Origin Airport and the month of the flight</u>**

Mapper :

```
class Mapper(api.Mapper):
        def map(self, context):
                data = context.value.split(",")
                # data[14] => delay time
                # data[16] => origin airport code
                key_value = data[16] + " " + data[1]
                # If there is no data we assume that the flight is on time
                if ((data[15] == 'NA') or (data[15] == '') or (data[15] == ' ')):
                        temp = 0.0
                elif (re.search('[a-zA-Z]', data[15])):
                        temp = -1.0
                else:
                        temp = float(data[15])

                if (temp > 0):
                        context.emit(key_value, 1)
                else:
                        context.emit(key_value, 0)
```

- Every mapper function will receive a single flight record as the input.

- Since, it is a csv file we will split the record based on the delimiter ‘,’.
- Now when we take a look at the data file, we can see that the 15th field is the delay time and the 17th field is the origin airport and the 2nd field is the month.
- The only difference between the first and second parts is the key – here the key is a combination of both the origin and the month.
- Now if the delay is greater than 0 that means that the flight has arrived late.
- Whenever there is a delay we omit the value 1 from the mapper along with the key.
- The key here is the Origin airport code.

Reducer:

- The reducer code is same as for part 1.
- Summing the delays and the total number of flights and finding the percentage of delayed flights.

Output File:

```
bigd29@whale:/home2/cosc6339/bigd29/source> hdfs
dfs -cat /bigd29/output/output6/part-r-00000
ABE 1   26.8588770865
ABE 3   26.3601532567
ABE 5   20.9489051095
ABE 7   29.2042042042
ABE 9   27.3692810458
ABI 1   17.8272980501
ABI 3   21.5714285714
ABI 5   20.6599713056
ABI 7   22.6388888889
ABI 9   16.1849710983
ABQ 1   36.6555113993
ABQ 3   40.5839861634
ABQ 5   38.8472398014
ABQ 7   43.1298760899
ABQ 9   31.0517008218
ABY 1   38.905775076
ABY 3   28.5294117647
ABY 5   27.4760383387
ABY 7   40.1898734177
ABY 9   42.4342105263
ACK 6   54.1899441341
```

- A part of the output file can be seen above.
- The percentage is taken as float.
- The key – origin airport and the month the flight has occured
- The location of the output file is **/bigd29/output/output29/part-r-00000** on the HDFS file system.
- If only one reducer is considered – all the computations are done by 1 reducer.
- If multiple reducers are considered then they form subdirectories **part-r-00001** and so on.

Challenges and Assumptions:

i) First, when the code is being written for the short list used for code development – the data had a few NA and missing values which have to be considered while writing the mapper code.

```
if ((data[15] == 'NA') or (data[15] == '') or (data[15] == ' ')):
        temp = 0.0
```

If the value is missing I have assumed that the flight has arrived on time because by default the flights are supposed to arrive by the estimated time.

ii) And after incorporating these conditions the developmental code yielded good results.

iii) While testing the code with the 21Mill records, an error occurred which stated that there is a string 'DepDelay' which cannot be converted to an Integer.

iv) Then checking for any strings using Regular Expressions eliminated this error.

```
elif (re.search('[a-zA-Z]', data[15])):
        temp = -1.0
```

v) DepDelay is assumed to be "Departure Delay", meaning the flight has arrived late. Hence, a value of -1.0 which means that the flight has not arrived on time has been considered.

vi) After normalizing the delay data into understandable format, the emitting of 1s and 0s is done.

Specifications of the System:

Whale Cluster :

i) Nodes:
50 Appro 1522H nodes (whale-001 to whale-057), each node with

- Two 2.2 GHz quad-core AMD Opteron processor (8 cores total)
- 16 GB main memory
- Gigabit Ethernet
- 4xDDR InfiniBand HCAs (not used at the moment)

ii) Network Interconnect

- 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL)
- Two 48 port HP GE switch

iii) Storage

- 4 TB NFS / home file system (shared with crill)
- 7 TB HDFS file system (using triple replication)

Pydoop in Python:

i) It is a python interface to Hadoop that allows MapReduce applications to be written in Python

ii) Pydoop offers :
- A rich HDFS API
- A map-reduce API that allows python record readers/writers, partitioner and combiners.
- No python method for creating your own InputFormat, but possible to include Java inputformats

iii) Python 2.7

iv) Hadoop 3.0.3

Execution of the code

i) File Systems:
- First, it is important to understand that there are 2 systems here
  Local Linux system
  HDFS system
- Now the source code files are stored in the local Linux system from where we run the job.
- The data file which contains the 21 Mill flights records and the output location will be on the HDFS system.

ii) Copying the source code file from the local Desktop to Remotely connected Linux system.
Scp <local-location> Username@Host: <Remote-Location>
Enter the password when prompted for it

```
scp sample.txt bigd29@129.7.243.176:/home2/
```

iii) Submitting a job:
pydoop submit --num-reducers 4 --upload-file-to-cache flights_delayed_origin.py flights_delayed_origin /cosc6339_hw1/flights-longlist/allflights.csv /bigd29/output/output1

The above command is used to run the job, which indicates the number of reducers used, name of the source file, hdfs location of the data file, hdfs location of the output directory

iv) Viewing the output:
hdfs dfs -cat /bigd29/output/output5/part-r-00000

The above command has the location of the output file.

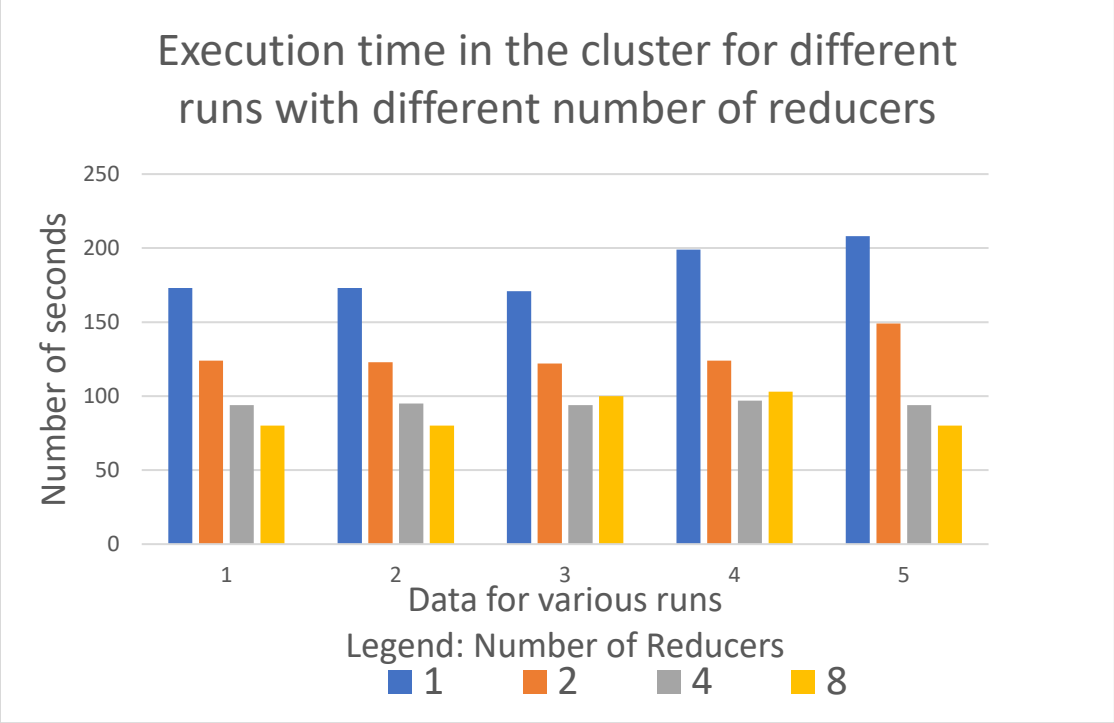
Results & Finding:

i) **Based on Origin Airport of the flight:**

Runs\No. Reducers	1	2	4	8
1	183	126	122	118
2	172	123	97	83
3	170	122	96	83
4	169	121	97	86
5	169	123	117	93

Units in seconds – Table showing the number of seconds the job ran for different runs with different reducers.

- We can see a pattern here – With increase in the number of reducers – there is a decrease in the execution time.
- The least time being 169, 121, 96 and 83 seconds for reducers 1,2,4,8 respectively.
- This might be due to a lesser number of keys (compared to part-2)– as they are easily completed when more number of reducers are used.



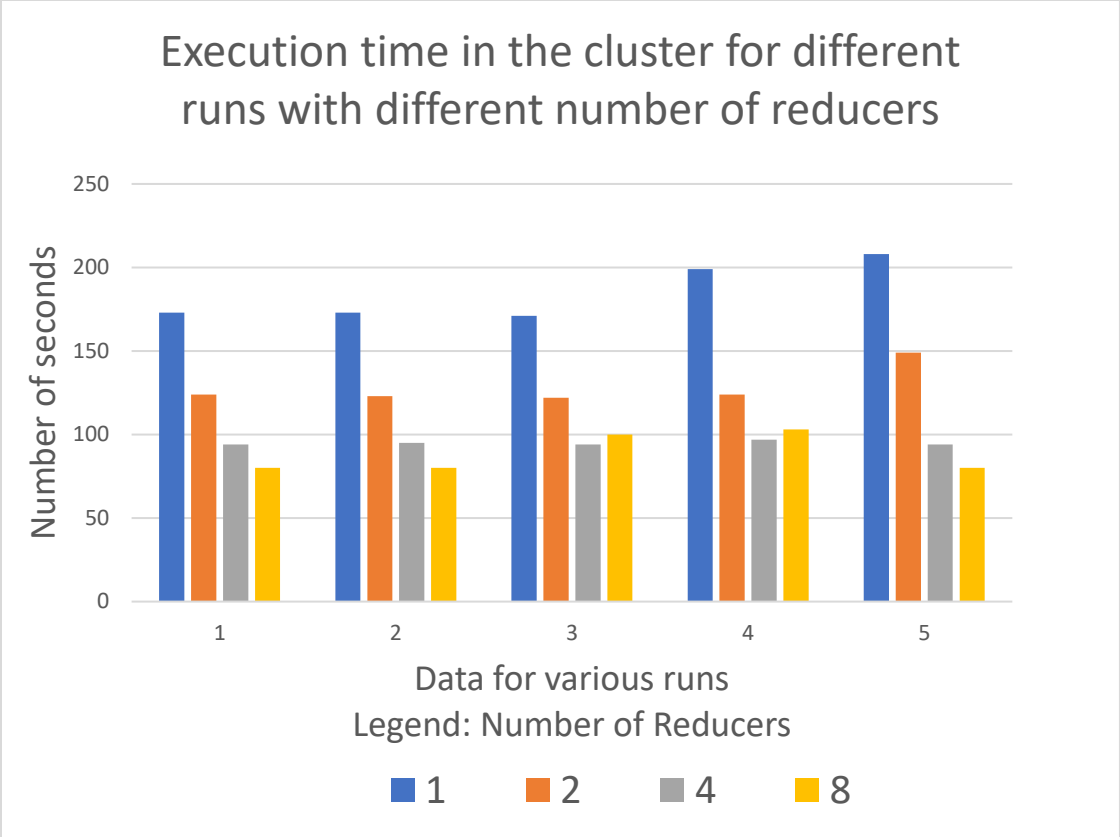
- From this table we can clearly see the trend of how the execution time decreases.
- We can also see a sudden drop when it goes from 1 reducer to 2 and then very slight decrease with increase in the number of reducers.
- I also checked with **Reducer number : 16,32 and 64** and the execution time is 78,76,77 respectively – we can see how less of a difference increasing the reducers will make.
- But when the measurements for the shorflights.csv is done, 23, 24, 24, 24, can be seen for reducers 1,2,4,8 respectively.
- This is due to the less amount of data available for this file, so we do not observe much difference. We can also see a increasing trend in such cases because of the less number of records.

ii) **Based on Origin Airport and month of the flight:**

Runs\No. Reducers	1	2	4	8
1	173	124	94	80
2	173	123	95	80
3	171	122	94	100
4	199	124	97	103
5	208	149	94	80

Units in seconds – Table showing the number of seconds the job ran for different runs with different reducers.

- We can see a pattern here – With increase in the number of reducers – there is a decrease in the execution time.
- The least time being 171, 122, 94 and 80 seconds for reducers 1,2,4,8 respectively.
- We can also see some inconsistent values like 208 and 199. This can be due to the load on the cluster when the job was being performed.



- From this table we can clearly see the trend of how the execution time decreases.
- We can also see a sudden drop when it goes from 1 reducer to 2 and then very slight decrease with increase in the number of reducers.
- We can conclude that there is a gradual decrease in time with the increase in the number of reducers. But the slope of the line is decreasing – meaning we will arrive at a point when the increase in the number of reducers will no longer contribute to the decrease in time.