1. Brief overview of data extraction and cleaning.

As mentioned in the project description in this project we were asked to work on the dataset from the US Bureau of Transportation Statistics (BTS). The dataset was hosted as an Amazon EBS volume snapshot with US Snapshot ID (Linux/Unix): snap-e1608d88 and size of 15 GB. Therefore, I have decided to utilize the AWS-EC2 to copy and extract the data. After extracting the data and selecting the necessary fields I have assembled all the data from different files to a single csv file and store it into AWS-S3 (shown in Fig. 1). I have followed the following steps as also shown in Fig. 1 to extract and prepare the data:

1. Create an EC2 instance: (Amazon Linux 2 AMI) (t2.medium) (SSD 20 GB).
2. Copy the EBS snapshot (snap-e1608d88) and create a volume from it.
3. Attach the EBS volume to the EC2 instance.
4. SSH to the EC2 and mount the EBS volume to the EC2 instance using following commands:
   sudo mkdir /data
   sudo mount /dev/xvdf /data
5. To efficiently extract the data I have prepared a python code (please see Appendix A) to go over the data folders and subfolders, open the zip files in each subfolders, read the csv file and combine all the data to a single csv file as an output. Also as I realized I don't need all the columns in the original data, to be more efficient, I have decided to collect only the following columns of the data: Year, Month, Origin, ArrTime, DepTime, Dest, ArrDelay, UniqueCarrier, DayOfWeek, DepDelay, and Cancelled. The python code is provided as Appendix A.
6. The prepared csv file collected from all the zip files then stored in a AWS-S3 bucket:
   - To connect the S3 and EC2 the IAM policy should be created and also both EC2 and S3 should be created in the same region.
   - Access Key for the bucket should be set in EC2 using the following command:
     aws configure
   - Then we can use the following command to copy the file from EC2 to S3
     aws s3 cp CleanedData.csv s3://cloudcoursecap/aviation/
7. Further cleaning the data: I observed that there are some null values in the data also I wanted to filter out the cancelled flights from the data but I realized that the part of the results provided to us in "Task 1 Example Solutions" to check our results were calculated without these filtrations. So my results for group 1 and question-1 of group 2 are based on no further cleaning but for group2 question 2, 3, 4 and group 3 questions I have further cleaned the data and omit the cancelled flights or any null data. I have decided to use pySpark queries: data rows with null values omitted from the dataset and also data rows with Cancelled column values equal to 1 are not considered to provide part of the results.

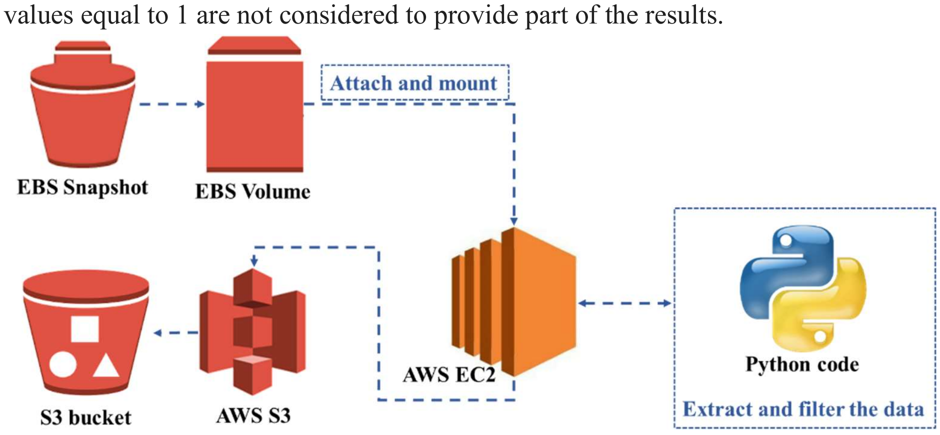


Fig. 1. Steps to extract the data from AWS-EBS snapshot to store the data in AWS-S3.

2. Overview of how the system is integrated.

The integration of the data extraction can be seen in Fig. 1 and the above explanation. After extracting the data following Fig.1 diagram and the above explanation I stored the data in AWS-S3. Then I have followed the following steps to answer the questions as also shown in Fig. 2 diagram:

1. Create an AWS-EMR (elastic MapReduce):
    - I have used **EMR** 6.0.0 with **Hadoop** 3.2.1 and **Spark** 2.4.4 using one master node and 2 cores.
    - I have also used the **bootstrap** option to install: pip, matplotlib, pandas and scipy.
2. On the EMR I created a **PySpark** jupyter.
3. Load the data from AWS-S3 to the PySpark.
4. For the questions which needed to be stored on **DynamoDB** first I have created a separate table for each in DynamoDB.
5. To efficiently store the queries results into the DynamoDB I have created an AWS **Lambda** function to automatically push to DynamoDB table as explained below.
6. I set the trigger for the Lambda function as creation of a file on a S3 bucket using **boto3**. So when in PySpark I obtain the result for a question, I stored the obtained result into S3 then automatically it will be pushed to the DynamoDB table using the Lambda function.
7. To connect S3, DynamoDB and CloudWatch to lambda function I also needed to create an IAM policy and a role to attach the policy to the Lambda function.
8. Code for the Lambda function is provided in Appendix B (The provided code is for group 2 question1, for the other questions a similar code was used but the name of table and the name of bucket as a trigger was different.
9. I also created a Cloudwatch to debug the system.

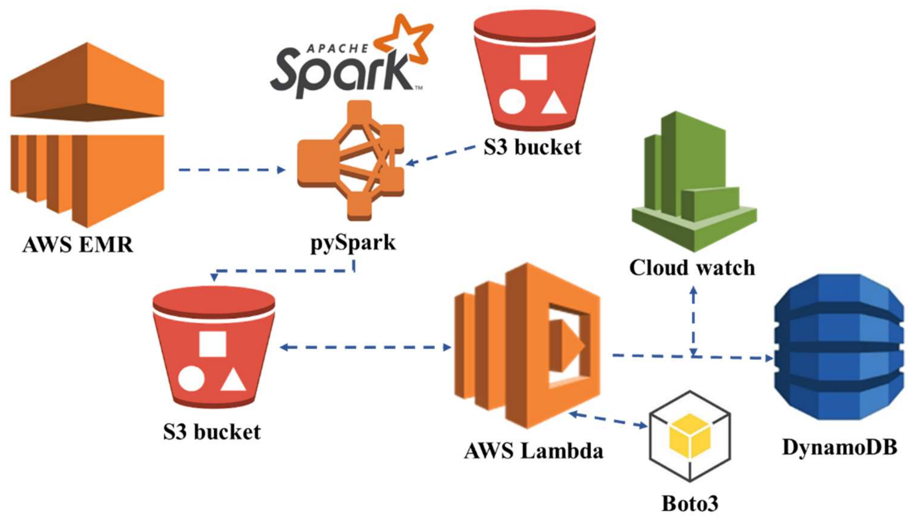


Fig. 2. Steps to answer the questions using PySpark and store in S3 and DynamoDB.

3. Approaches and algorithms used to answer each question.

To answer the questions I have used python and PySpark on elastic map-reduce using PySpark SQL by utilizing commands such as: alias, groupby, filter, select, collect, count, union, join, sum, avg, and where. I have answered the question for group2 and group3 also with utilizing PySpark SQL, however, as an additional practice after storing the results in DynamoDB I also inquired the results in DynamoDB. Following I have shown my inquiries for each question (You can see my entire Jupyter notebook in Appendix C):

Group 1

Question 1

```
popularityTable = df.groupBy("Dest").count().union(df.groupBy("Origin").count())\
                        .groupBy("Dest").sum('count').orderBy('sum(count)', ascending=False)
popularityTable.show(10)
```

Question 2

```
df.groupBy("UniqueCarrier").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True).show(10)
```

Question 3

```
df.groupBy("DayOfWeek").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True).show()
```

Group 2

Question 1

```
df3 = df2.groupBy("Origin", "UniqueCarrier").avg('DepDelay')\
          .orderBy(['Origin', 'avg(DepDelay)'], ascending=[True,True])
#select just 10 carriers in decreasing order for each airport
window = Window.partitionBy(df3["Origin"]).orderBy(df3['avg(DepDelay)'])
df3 = df3.select(col('*'), row_number().over(window).alias('row_number'))\
          .where(col('row_number') <= 10).orderBy(["Origin","avg(DepDelay)"])
```

Then inquiry for each airport as for example:

```
df3.filter(df.Origin == "CMI")
```

Question 2

```
df3 = df2.groupBy("Origin", "Dest").avg('DepDelay')\
      .orderBy(['Origin', 'avg(DepDelay)'], ascending=[True,True])
#select just 10 destination for each airport
window = Window.partitionBy(df3["Origin"]).orderBy(df3['avg(DepDelay)'])
df3 = df3.select(col('*'), row_number().over(window).alias('row_number'))\
      .where(col('row_number') <= 10).orderBy(["Origin","avg(DepDelay)"])
```

Then inquiry for each airport as for example:

```
df3.filter(df.Origin == "CMI")
```

Question 3

```
df3 = df2.groupBy("Origin", "Dest", "UniqueCarrier").avg('ArrDelay')\
      .orderBy(['Origin', 'Dest', 'avg(ArrDelay)'], ascending=[True,True,True])
```

Then inquiry for each source-destination pair X-Y as for example:

```
df3.filter(df3.Origin == "CMI").filter(df3.Dest == "ORD").show(10)
```

Question 4

```
df3 = df2.groupBy("Origin", "Dest").avg('ArrDelay').orderBy("Origin")
```

Then inquiry for each source-destination pair X-Y as for example:

```
df3.filter(df3.Origin == "CMI").filter(df3.Dest == "ORD").show()
```

For group 3 question 2, first I selected 2008 data, second I divided the data into before and after 12:00 PM and then join the data on X destination and Y origin. To save the data into DynamoDB table I considered to save only the provided X-Y-Z routes. For each X-Y-Z I saved the top 10 lowest arrival delay for each days of the month. Please see the below code.

Group 3

Question 1

```
popularityTable = df.groupBy("Dest").count().union(df.groupBy("Origin").count())\
                        .groupBy("Dest").sum('count').orderBy('sum(count)', ascending=False)
popularitlList = popularityTable.select('sum(count)')
#Convert panda dataframe to python list
p2=[list(row)[0] for row in popularitlList.collect()]
```

```
# Histogram of the popularity with normal distribution graph
plt.clf()
p3 = np.log(p2)
plt.hist(p3, color = 'blue', edgecolor = 'black', bins = 100, density=True)
pdf = stats.norm.pdf(p3, np.mean(p3), np.std(p3))
plt.plot(p3, pdf, color = 'red')
plt.ylabel("Freq.")
plt.xlabel("Log(popularity)")
plt.grid(True)
plt.show()
```

```
%matplot plt
plt.show()
#CCDF graph of the popularity
plt.clf()
yvals=np.arange(len(p2))/float(len(p2)-1)
plt.loglog(p2,yvals)
plt.ylabel("CCDF")
plt.xlabel("X")
plt.grid(True)
plt.show()
```

```
%matplot plt
plt.show()
```

Question 2

```
# Select 2008 data
df2008 = df2.filter(df2.Year == "2008").sort(["month", "DayofMonth"])
```

```
# Divide data for Afte and Before 12:00 PM
df_X = df2008.filter(df2008.DepTime < 1200.0)
df_Y = df2008.filter(df2008.DepTime > 1200.0)
```

```
from functools import reduce
oldColumns = df_X.schema.names
newColumns = ["Year_X", "Month_X", "Origin_X",
              "ArrTime_X", "DepTime_X", "Dest_X",
              "ArrDelay_X", "UniqueCarrier_X",
              "DayOfWeek_X", "DayofMonth_X",
              "DepDelay_X", "Cancelled_X"]
#Find all eligible pairs for X origin
df_X = reduce(lambda df_X, idx: df_X.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_X)
```

```
oldColumns = df_Y.schema.names
newColumns = ["Year_Y", "Month_Y", "Origin_Y",
              "ArrTime_Y", "DepTime_Y", "Dest_Y",
              "ArrDelay_Y", "UniqueCarrier_Y",
              "DayOfWeek_Y", "DayofMonth_Y",
              "DepDelay_Y", "Cancelled_Y"]
#Find all eligible pairs for Y origin
df_Y = reduce(lambda df_Y, idx: df_Y.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_Y)
```

```
import pyspark.sql.functions as f
# Combine df_X and df_Y to satisfy the X-Y-Z conditions
df4 = df_Y.alias('l').join(df_X.alias('r'), (f.col('l.Origin_Y') == f.col('r.Dest_X')) \
                          & (f.col('l.Month_Y') == f.col('r.Month_X')) \
                          & (f.col('l.DayofMonth_Y') == 2 + f.col('r.DayofMonth_X')))\
        .select('r.DayofMonth_X','r.Month_X','r.Year_X', 'r.DepTime_X', 'r.Origin_X', 'r.Dest_X', 'r.ArrDelay_X', 'l.Dayc
df4 = df4.withColumn('TotalArrivalDelay', df4.ArrDelay_X + df4.ArrDelay_Y)
```

Then inquiry for each X-Y-Z and day/month as for example:

```
df4.filter(df4.Origin_X == "CMI").filter(df4.Dest_X == "ORD")\
   .filter(df4.Dest_Y == "LAX").filter(df4.Month_X == 3)\
   .filter(df4.DayofMonth_X == 4).sort(["TotalArrivalDelay"]).show(1)
```

What are the results of each question? Use only the provided subset for questions from Group 2 and Question 3.2.

Group 1

Question 1

Dest	sum(count)
ORD	12449354
ATL	11540422
DFW	10799303
LAX	7723596
PHX	6585534
DEN	6273787
DTW	5636622
IAH	5480734
MSP	5199213
SFO	5171023

Question 2

UniqueCarrier	avg(ArrDelay)
HA	-1.01180434574519
AQ	1.1569234424812056
PS	1.4506385127822803
ML (1)	4.747609195734892
PA (1)	5.3224309999287875
F9	5.465881148819851
NW	5.557783392671835
WN	5.5607742598815735
OO	5.736312463662878
9E	5.8671846616957595

Question 3

DayOfWeek	avg(ArrDelay)
6	4.301669926076596
2	5.990458841319885
7	6.613280292442754
1	6.716102802585582
3	7.203656394670348
4	9.094441008336657
5	9.721032337585571

Group 2

Question 1

Origin	UniqueCarrier	avg(DepDelay)
CMI	OH	0.6116264687693259
CMI	US	2.033047346679377
CMI	TW	4.120615384615385
CMI	PI	4.455628350208458
CMI	DH	6.027888446215139
CMI	EV	6.665137614678899
CMI	MQ	8.016004886988393

Origin	UniqueCarrier	avg(DepDelay)
BWI	F9	0.7562437562437563
BWI	PA (1)	4.761904761904762
BWI	CO	5.179340976854271
BWI	YV	5.496503496503497
BWI	NW	5.705573031597727
BWI	AA	6.002851840115884
BWI	9E	7.239805825242718
BWI	US	7.494395794023255
BWI	DL	7.676822368501101
BWI	UA	7.737921397819683

Origin	UniqueCarrier	avg(DepDelay)
MIA	9E	-3.0
MIA	EV	1.2026431718061674
MIA	TZ	1.782243551289742
MIA	XE	1.8731909028256375
MIA	PA (1)	4.20000428045544
MIA	NW	4.501665523660233
MIA	US	6.090665809518826
MIA	UA	6.869731753577851
MIA	ML (1)	7.504550050556118
MIA	FL	8.565107458912768

Origin	UniqueCarrier	avg(DepDelay)
LAX	MQ	2.407221858260434
LAX	OO	4.2219592877139975
LAX	FL	4.725127379994636
LAX	TZ	4.763940985246312
LAX	PS	4.860337041524397
LAX	NW	5.11955065127997
LAX	F9	5.729155372438469
LAX	HA	5.813645621181263
LAX	YV	6.024156085475379
LAX	US	6.746395368371022

Origin	UniqueCarrier	avg(DepDelay)
IAH	NW	3.5637106119971302
IAH	PA (1)	3.9847272727272727
IAH	PI	3.9886668654935877
IAH	US	5.060267573407907
IAH	F9	5.545243619489559
IAH	AA	5.703959137557669
IAH	TW	6.048777413662718
IAH	WN	6.231133355443664
IAH	OO	6.58795822240426
IAH	MQ	6.7129735935706085

Origin	UniqueCarrier	avg(DepDelay)
SFO	TZ	3.952415634862831
SFO	MQ	4.853923777799549
SFO	F9	5.162444663059518
SFO	PA (1)	5.28761165961448
SFO	NW	5.757805769125906
SFO	PS	6.303518700787402
SFO	DL	6.562729888421325
SFO	CO	7.0830491940353975
SFO	US	7.527510076713042
SFO	TW	7.79488255033557

Group 2

Question 2

Origin	Dest	avg(DepDelay)
CMI	ABI	-7.0
CMI	PIT	1.1024305555555556
CMI	CVG	1.8947616800377536
CMI	DAY	3.116235294117647
CMI	STL	3.981673306772908
CMI	PIA	4.591891891891892
CMI	DFW	5.944142746314973
CMI	ATL	6.665137614678899
CMI	ORD	8.194098143236074

Origin	Dest	avg(DepDelay)
BWI	SAV	-7.0
BWI	MLB	1.155367231638418
BWI	DAB	1.4695945945945945
BWI	SRQ	1.5884838880084522
BWI	IAD	1.7909407665505226
BWI	UCA	3.6541698546289214
BWI	CHO	3.744927536231884
BWI	GSP	4.197686645636172
BWI	SJU	4.44465842286641
BWI	OAJ	4.471111111111111

Origin	Dest	avg(DepDelay)
MIA	SHV	0.0
MIA	BUF	1.0
MIA	SAN	1.710382513661202
MIA	SLC	2.5371900826446283
MIA	HOU	2.912199124726477
MIA	ISP	3.647398843930636
MIA	MEM	3.7451066224751424
MIA	PSE	3.975845410628019
MIA	TLH	4.2614844746916205
MIA	MCI	4.612244897959184

Origin	Dest	avg(DepDelay)
LAX	SDF	-16.0
LAX	IDA	-7.0
LAX	DRO	-6.0
LAX	RSW	-3.0
LAX	LAX	-2.0
LAX	BZN	-0.7272727272727273
LAX	MAF	0.0
LAX	PIH	0.0
LAX	IYK	1.2698247440569148
LAX	MFE	1.3764705882352941

Origin	Dest	avg(DepDelay)
IAH	MSN	-2.0
IAH	AGS	-0.6187904967602592
IAH	MLI	-0.5
IAH	EFD	1.8877082136703045
IAH	HOU	2.172036985149902
IAH	JAC	2.570588235294118
IAH	MTJ	2.9501569858712715
IAH	RNO	3.22158438576349
IAH	BPT	3.5995325282430852
IAH	VCT	3.6119087837837838

Origin	Dest	avg(DepDelay)
SFO	SDF	-10.0
SFO	MSO	-4.0
SFO	PIH	-3.0
SFO	LGA	-1.7575757575757576
SFO	PIE	-1.3410404624277457
SFO	OAK	-0.813200498132005
SFO	FAR	0.0
SFO	BNA	2.425966447848286
SFO	MEM	3.302482299752623
SFO	SCK	4.0

Group 2

Question 3

Origin	Dest	UniqueCarrier	avg(ArrDelay)
CMI	ORD	MQ	10.14366290643663

Origin	Dest	UniqueCarrier	avg(ArrDelay)
IND	CMH	CO	-2.54585456229736
IND	CMH	AA	5.5
IND	CMH	HP	5.697254901960784
IND	CMH	NW	5.7615384615384615
IND	CMH	US	6.878469415251954
IND	CMH	DL	10.6875
IND	CMH	EA	10.813084112149532

Origin	Dest	UniqueCarrier	avg(ArrDelay)
DFW	IAH	PA (1)	-1.5964912280701755
DFW	IAH	EV	5.0925133689839575
DFW	IAH	UA	5.414201183431953
DFW	IAH	CO	6.493731644930054
DFW	IAH	OO	7.564007421150278
DFW	IAH	XE	8.094294547498595
DFW	IAH	AA	8.381228324333817
DFW	IAH	DL	8.598509052183173
DFW	IAH	MQ	9.103211009174313

Origin	Dest	UniqueCarrier	avg(ArrDelay)
LAX	SFO	TZ	-7.619047619047619
LAX	SFO	PS	-2.1463414634146343
LAX	SFO	F9	-2.028685790527018
LAX	SFO	EV	6.964630225080386
LAX	SFO	AA	7.386793490213328
LAX	SFO	MQ	7.8077634011090575
LAX	SFO	US	7.964721980345814
LAX	SFO	WN	8.79205149734117
LAX	SFO	CO	9.354782608695652
LAX	SFO	NW	9.84878587196468

Origin	Dest	UniqueCarrier	avg(ArrDelay)
JFK	LAX	UA	3.313874383174436
JFK	LAX	HP	6.680599369085174
JFK	LAX	AA	6.90372453707467
JFK	LAX	DL	7.934460351304701
JFK	LAX	PA (1)	11.019443694301918
JFK	LAX	TW	11.702008082849204

Origin	Dest	UniqueCarrier	avg(ArrDelay)
ATL	PHX	FL	4.552631578947368
ATL	PHX	US	6.28811524609844
ATL	PHX	HP	8.481436314363144
ATL	PHX	EA	8.95357142857143
ATL	PHX	DL	9.808275435290147

Group 2

Question 4

Origin	Dest	avg(ArrDelay)
CMI	ORD	10.14366290643663

Origin	Dest	avg(ArrDelay)
IND	CMH	2.89990366088632

Origin	Dest	avg(ArrDelay)
DFW	IAH	7.654442525768608

Origin	Dest	avg(ArrDelay)
LAX	SFO	9.589282731105238

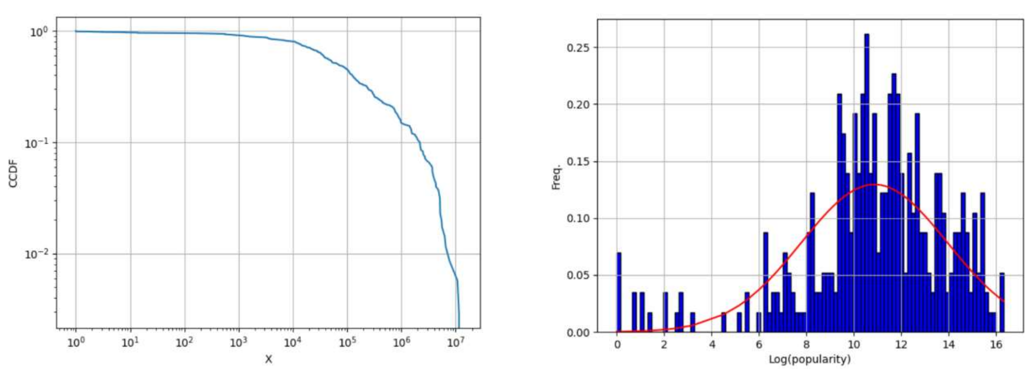
Origin	Dest	avg(ArrDelay)
JFK	LAX	6.635119155270517

Origin	Dest	avg(ArrDelay)
ATL	PHX	9.021341881513989

Group 3

Question 1

Fig. 3 (a) shows the log-log CCDF of the airports popularity distribution. Also Fig. 3(b) shows log histogram of airports popularity distribution along with fitted log normal distribution. As it can be seen as the CCDF does not follow a straight line, we can reject the hypothesis that the data was drawn from a power-law distribution. Also as it can be seen in Fig. 3 (b) a simple log-normal distribution on the airports popularity histogram is well fitted the data so we can see that the distribution is close to the normal distribution, however, it can be well-fitted as a t-distribution.



(a) **(b)**

Fig. 3. Popularity distribution of airports: (a) CCDF; (b) Histogram together with normal distribution.

Question 2

Group 3

Question 2

CMI → ORD → LAX

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
4	3	2008	710.0	CMI	ORD	-14.0	6	3	2008	1952.0	ORD	LAX	-24.0	-38.0

JAX → DFW → CRP

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
9	9	2008	722.0	JAX	DFW	1.0	11	9	2008	1648.0	DFW	CRP	-7.0	-6.0

SLC → BFL → LAX

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
1	4	2008	1101.0	SLC	BFL	12.0	3	4	2008	1509.0	BFL	LAX	6.0	18.0

LAX → SFO → PHX

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
12	7	2008	650.0	LAX	SFO	-13.0	14	7	2008	1916.0	SFO	PHX	-19.0	-32.0

DFW → ORD → DFW

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
10	6	2008	658.0	DFW	ORD	-21.0	12	6	2008	1650.0	ORD	DFW	-10.0	-31.0

LAX → ORD → JFK

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
1	1	2008	700.0	LAX	ORD	1.0	3	1	2008	1853.0	ORD	JFK	-7.0	-6.0

4. What system- or application-level optimizations (if any) did you employ?

- First of all to reduce the size of the data I explored the questions and selected the necessary fields required to answer the questions, hence, the size of the data drastically reduced. I have also assembled all the data file into a single file and then a single database.
- To make the queries faster I employed the AWS-EMR (elastic map reduce) which is kind of parallelism and then using PySpark on it.
- To obtain the results using PySpark I used the In-Memory Computation capability of Spark which makes the computation faster. It means PySpark does not perform the inquiry till reaching commands like: show.
- I used automatic capability of AWS-Lambda function. I created a Lambda function and set its trigger on AWS-S3 buckets as whenever I store the data on a specific folder in AWS-S3 bucket my lambda function automatically read the data and push it into the DynamoDB tables.

5. Give your opinion about whether the results make sense and are useful in any way.

Yes it is interesting to me. I did some researches for example according to the ClaimCompass the list of the top 10 most popular airports in the USA in 2020 was as the following:

ATL, LAX, ORD, DFW, DEN, JFK, SFO, LAS, SEA, CLT

You can compare it with our results for 2008:

ORD, ATL, DFW, LAX, PHX, DEN, DTW, IAH, MSP, SFO

Also, according to the AFAR data Hawaiian Airlines: (87.4%) is the highest scored U.S.-based carriers in OAG's most recent report. In our data also (Group 1 question 2) HA: -1.01 was the top airline by on-time arrival performance. In addition, I think especially Group 3 question 2 can be very useful to find the possible route having two destination and specific time frame.

Appendix A

Python code to extract the data

```
"""
Created on Wed Jun 10 20:46:30 2020

@author: Bahman
"""
import glob
import os
import zipfile
import pandas as pd

col_names =  ['Year',
              'Month',
              'Origin',
              'ArrTime',
              'DepTime',
              'Dest',
              'ArrDelay',
              'UniqueCarrier',
              'DayOfWeek',
              'DepDelay',
              'Cancelled']

my_df  = pd.DataFrame(columns = col_names)
my_df.to_csv("CleanedData.csv", encoding='utf-8', index=False)

for dirpath, dirnames, filenames in os.walk("/data/aviation/airline_ontime/"):

    for name in glob.glob(dirpath + '/*.zip'):
        print(name)
        base = os.path.basename(name)
        filename = os.path.splitext(base)[0]

        datadirectory = dirpath + '/'
        dataFile = filename
        archive = '.'.join([dataFile, 'zip'])
        fullpath = ''.join([datadirectory, archive])
        csv_file = '.'.join([dataFile, 'csv'])

        filehandle = open(fullpath, 'rb')
        zfile = zipfile.ZipFile(filehandle)
        data = pd.read_csv(zfile.open(csv_file))
        data = data[col_names]
        data.to_csv("CleanedData.csv", encoding='utf-8', index=False, mode='a', header=False)
```

Appendix B

Sample of Lambda function code to push the data automatically from AWS-S3 to a Dynamo-DB table.

```
lambda_function ×

#Bahman
import boto3

s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

def csv_reader(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    obj = s3.get_object(Bucket=bucket, Key=key)

    rows = obj['Body'].read().split('\n')

    table = dynamodb.Table('group2_1')

    savedKey = {}
    DepDelay = {}

    with table.batch_writer() as batch:
        for row in rows:
            splited = row.split(',')
            if len(splited) < 2:
                continue
            if splited[0] in savedKey:
                savedKey[splited[0]].append(splited[1])
                DepDelay[splited[0]].append(splited[2])
            else:
                savedKey[splited[0]] = [splited[1]]
                DepDelay[splited[0]] = [splited[2]]


        for item in  savedKey:
            batch.put_item(Item={
                'Origin' : item,
                'UniqueCarrier' : savedKey[item],
                'avg(DepDelay)' : DepDelay[item]
            })
```

Appendix C

The complete PySpark code to answer the question and store the data into S3.

```
from pyspark.sql import functions as F
from pyspark.sql.types import FloatType
from pyspark.sql.types import IntegerType
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, col, rank
```

```
input_bucket = 's3://cloudcoursecap'
input_path = '/aviation/CleanedData.csv'
df = spark.read.csv(input_bucket + input_path, header=True)
df.show()
```

```
df = df.withColumn("ArrDelay",df["ArrDelay"].cast(FloatType()))
df = df.withColumn("DepDelay",df["DepDelay"].cast(FloatType()))
df = df.withColumn("DayOfWeek",df["DayOfWeek"].cast(IntegerType()))
df = df.withColumn("DayofMonth",df["DayofMonth"].cast(IntegerType()))
df = df.withColumn("Month",df["Month"].cast(IntegerType()))
df.count()
```

```
#Remove null data fields
#df = df.filter(df.ArrDelay.isNotNull())
#df2 = df.filter(df.DepDelay.isNotNull())
#omit cancelled flights
#df2 = df2.filter(df.Cancelled == "0.0")
df2 = df
```

```
#Group 1
# 1. Rank the top 10 most popular airports by numbers of flights to/from the airport.
popularityTable = df.groupBy("Dest").count().union(df.groupBy("Origin").count())\
                    .groupBy("Dest").sum('count').orderBy('sum(count)', ascending=False)
popularityTable.show(10)
```

```
#Group 1
# 2. Rank the top 10 airlines by on-time arrival performance.
df.groupBy("UniqueCarrier").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True).show(10)
```

```
#Group 1
# 3. Rank the days of the week by on-time arrival performance.
df.groupBy("DayOfWeek").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True).show()
```

```
#Group 2
# 1. For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.
df3 = df2.groupBy("Origin", "UniqueCarrier").avg('DepDelay')\
        .orderBy(['Origin', 'avg(DepDelay)'], ascending=[True,True])
#select just 10 carriers in decreasing order for each airport
window = Window.partitionBy(df3["Origin"]).orderBy(df3['avg(DepDelay)'])
df3 = df3.select(col('*'), row_number().over(window).alias('row_number'))\
        .where(col('row_number') <= 10).orderBy(["Origin","avg(DepDelay)"])
#df3.show(df3.count(), False)
df3.show()
```

```
df3.filter(df.Origin == "CMI").show(10)
```

```
df3.filter(df.Origin == "BWI").show(10)
```

```
df3.filter(df.Origin == "MIA").show(10)
```

```
df3.filter(df.Origin == "LAX").show(10)
```

```
df3.filter(df.Origin == "IAH").show(10)
```

```
df3.filter(df.Origin == "SFO").show(10)
```

```
# Writing the results to S3 and automatically in DynamoDB
input_path = '/Result/DynamoDB/Group2/1/'
df3.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True)
```

```
df = df.filter(df.ArrDelay.isNotNull())
df2 = df.filter(df.DepDelay.isNotNull())
#Group 2
# 3. For each source-destination pair X-Y, rank the top-10 carriers in decreasing
#    order of on-time arrival performance at Y from X.
df3 = df2.groupBy("Origin", "Dest", "UniqueCarrier").avg('ArrDelay')\
      .orderBy(['Origin', 'Dest', 'avg(ArrDelay)'], ascending=[True,True,True])
#df3.show(df3.count(),False)
df3.show()
```

```
df3.filter(df3.Origin == "CMI").filter(df3.Dest == "ORD").show(10)
```

```
df3.filter(df3.Origin == "IND").filter(df3.Dest == "CMH").show(10)
```

```
df3.filter(df3.Origin == "DFW").filter(df3.Dest == "IAH").show(10)
```

```
df3.filter(df3.Origin == "LAX").filter(df3.Dest == "SFO").show(10)
```

```
df3.filter(df3.Origin == "JFK").filter(df3.Dest == "LAX").show(10)
```

```
df3.filter(df3.Origin == "ATL").filter(df3.Dest == "PHX").show(10)
```

```
input_path = '/Result/DynamoDB/Group2/3/'
df3.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True)
```

```
#Group 2
# 4. For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.
df3 = df2.groupBy("Origin", "Dest").avg('ArrDelay').orderBy("Origin")
df3.show()
```

```
df3.filter(df3.Origin == "CMI").filter(df3.Dest == "ORD").show()
```

```
df3.filter(df3.Origin == "IND").filter(df3.Dest == "CMH").show()
```

```
df3.filter(df3.Origin == "DFW").filter(df3.Dest == "IAH").show()
```

```
df3.filter(df3.Origin == "LAX").filter(df3.Dest == "SFO").show()
```

```
df3.filter(df3.Origin == "JFK").filter(df3.Dest == "LAX").show()
```

```
df3.filter(df3.Origin == "ATL").filter(df3.Dest == "PHX").show()
```

```
input_path = '/Result/DynamoDB/Group2/4/'
df3.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True)
```

```
# Group 3:
#1. Does the popularity distribution of airports follow a Zipf distribution?
#If not, what distribution does it follow?
popularitlList = popularityTable.select('sum(count)')
#Convert panda dataframe to python list
p2=[list(row)[0] for row in popularitlList.collect()]
```

```
import numpy as np
import scipy
import matplotlib
import matplotlib.pyplot as plt
import scipy.stats as stats

# Histogram of the popularity with normal distribution graph
plt.clf()
p3 = np.log(p2)
plt.hist(p3, color = 'blue', edgecolor = 'black', bins = 100, density=True)
pdf = stats.norm.pdf(p3, np.mean(p3), np.std(p3))
plt.plot(p3, pdf, color = 'red')
plt.ylabel("Freq.")
plt.xlabel("Log(popularity)")
plt.grid(True)
plt.show()
```

```
%matplot plt
plt.show()
```

```
#CCDF graph of the popularity
plt.clf()
yvals=np.arange(len(p2))/float(len(p2)-1)
plt.loglog(p2,yvals)
plt.ylabel("CCDF")
plt.xlabel("X")
plt.grid(True)
plt.show()
```

```
%matplot plt
plt.show()
```

```
# Group 3:
# Tom wants to travel from airport X to airport Z.
# Select 2008 data
df2008 = df2.filter(df2.Year == "2008").sort(["month", "DayofMonth"])
df2008.show()
```

```
# Divide data for Afte and Before 12:00 PM
df_X = df2008.filter(df2008.DepTime < 1200.0)
df_Y = df2008.filter(df2008.DepTime > 1200.0)
```

```
from functools import reduce
oldColumns = df_X.schema.names
newColumns = ["Year_X", "Month_X", "Origin_X",
              "ArrTime_X", "DepTime_X", "Dest_X",
              "ArrDelay_X", "UniqueCarrier_X",
              "DayOfWeek_X", "DayofMonth_X",
              "DepDelay_X", "Cancelled_X"]
#Find all eligible pairs for X origin
df_X = reduce(lambda df_X, idx: df_X.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_X)
```

```
oldColumns = df_Y.schema.names
newColumns = ["Year_Y", "Month_Y", "Origin_Y",
              "ArrTime_Y", "DepTime_Y", "Dest_Y",
              "ArrDelay_Y", "UniqueCarrier_Y",
              "DayOfWeek_Y", "DayofMonth_Y",
              "DepDelay_Y", "Cancelled_Y"]
#Find all eligible pairs for Y origin
df_Y = reduce(lambda df_Y, idx: df_Y.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_Y)
```

```
import pyspark.sql.functions as f
# Combine df_X and df_Y to satisfy the X-Y-Z conditions
df4 = df_Y.alias('l').join(df_X.alias('r'), (f.col('l.Origin_Y') == f.col('r.Dest_X')) \
                           & (f.col('l.Month_Y') == f.col('r.Month_X')) \
                           & (f.col('l.DayofMonth_Y') == 2 + f.col('r.DayofMonth_X')))\
          .select('r.DayofMonth_X','r.Month_X','r.Year_X', 'r.DepTime_X', 'r.Origin_X', 'r.Dest_X', 'r.ArrDelay_X', 'l.DayofMo
df4 = df4.withColumn('TotalArrivalDelay', df4.ArrDelay_X + df4.ArrDelay_Y)
df4.show()
```

```
# CMI → ORD → LAX
df4.filter(df4.Origin_X == "CMI").filter(df4.Dest_X == "ORD")\
   .filter(df4.Dest_Y == "LAX").filter(df4.Month_X == 3)\
   .filter(df4.DayofMonth_X == 4).sort(["TotalArrivalDelay"]).show(1)
```

```
# JAX → DFW → CRP
df4.filter(df4.Origin_X == "JAX").filter(df4.Dest_X == "DFW")\
   .filter(df4.Dest_Y == "CRP").filter(df4.Month_X == 9)\
   .filter(df4.DayofMonth_X == 9).sort(["TotalArrivalDelay"]).show(1)
```

```
# SLC → BFL → LAX
df4.filter(df4.Origin_X == "SLC").filter(df4.Dest_X == "BFL")\
   .filter(df4.Dest_Y == "LAX").filter(df4.Month_X == 4)\
   .filter(df4.DayofMonth_X == 1).sort(["TotalArrivalDelay"]).show(1)
```

```
# LAX → SFO → PHX
df4.filter(df4.Origin_X == "LAX").filter(df4.Dest_X == "SFO")\
   .filter(df4.Dest_Y == "PHX").filter(df4.Month_X == 7)\
   .filter(df4.DayofMonth_X == 12).sort(["TotalArrivalDelay"]).show(1)
```

```
# DFW → ORD → DFW
df4.filter(df4.Origin_X == "DFW").filter(df4.Dest_X == "ORD")\
   .filter(df4.Dest_Y == "DFW").filter(df4.Month_X == 6)\
   .filter(df4.DayofMonth_X == 10).sort(["TotalArrivalDelay"]).show(1)
```

```
# LAX → ORD → JFK
df4.filter(df4.Origin_X == "LAX").filter(df4.Dest_X == "ORD")\
   .filter(df4.Dest_Y == "JFK").filter(df4.Month_X == 1)\
   .filter(df4.DayofMonth_X == 1).sort(["TotalArrivalDelay"]).show(1)
```

```
input_path = '/Result/DynamoDB_/Group3/2/'
# CMI → ORD → LAX
#Filter data with Origin X as: "CMI" Y as "ORD" and Z as "LAX"
df5 = df4.filter((df4.Origin_X == "CMI") & (df4.Dest_X == "ORD") & (df4.Dest_Y == "LAX"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```

```
# JAX → DFW → CRP
#Filter data with Origin X as: "JAX" Y as "DFW" and Z as "CRP"
df5 = df4.filter((df4.Origin_X == "JAX") & (df4.Dest_X == "DFW") & (df4.Dest_Y == "CRP"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```

```
# SLC → BFL → LAX
#Filter data with Origin X as: "SLC" Y as "BFL" and Z as "LAX"
df5 = df4.filter((df4.Origin_X == "SLC") & (df4.Dest_X == "BFL") & (df4.Dest_Y == "LAX"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```

```
# LAX → SFO → PHX
#Filter data with Origin X as: "LAX" Y as "SFO" and Z as "PHX"
df5 = df4.filter((df4.Origin_X == "LAX") & (df4.Dest_X == "SFO") & (df4.Dest_Y == "PHX"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```

```
# DFW → ORD → DFW
#Filter data with Origin X as: "DFW" Y as "ORD" and Z as "DFW"
df5 = df4.filter((df4.Origin_X == "DFW") & (df4.Dest_X == "ORD") & (df4.Dest_Y == "DFW"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```

```
# LAX → ORD → JFK
#Filter data with Origin X as: "LAX" Y as "ORD" and Z as "JFK"
df5 = df4.filter((df4.Origin_X == "LAX") & (df4.Dest_X == "ORD") & (df4.Dest_Y == "JFK"))
#select best 10 X-Y-Z for each day of each month
window = Window.partitionBy(df5["Month_X"]).partitionBy(df5["DayofMonth_X"]).orderBy(df5['TotalArrivalDelay'])
df5 = df5.select(col('*'), row_number().over(window).alias('row_number'))\
         .where(col('row_number') <= 10)
#save in S3 and DynamoDB(via Lambda)
df5.coalesce(1).write.save(input_bucket + input_path, format='csv', header=True, mode="overwrite")
```